DOCUMENT RESUME

ED 247 242		TM 840 393

AUTHOR	Quan, Baldwin; And Others
TITLE	Microcomputer Network for Computerized Adaptive Testing (CAT): Program Listing.
INSTITUTION	Navy Personnel Research and Development Center, San Diego, Calif.
REPORT NO	NPRDC-TR-84-33-Suppl
PUB DATE	Mar 84
NOTE	397p.; For related document, see TM 840 392.
PUB TYPE	Audiovisual Materials (100)
EDRS PRICE	MF01/PC16 Plus Postage.
DESCRIPTORS	*Adaptive Testing; Bayesian Statistics; *Computer Assisted Testing; *Computer Software; Microcomputers; Reference Materials
IDENTIFIERS	PASCAL Programing Language

ABSTRACT

This program listing is a supplement to the Microcomputer Network for Computerized Adaptive Testing (CAT). The driver textfile program allows access to major subprograms of the CAT project. The test administration textfile program gives examinees a prescribed set of subtests. The parameter management textfile program establishes a file containing the testing strategies and parameters given to examinees. The test manager textfile program provides file maintenance of tests; allows creation of new tests and editing of test questions; lists tests to file, console, or printer for verification; and writes subtests from the hard disk to a floppy disk. The examinee manager textfile program allows access to and editing of examinee records; lists examinee's test scores; removes or modifies an examinee record; controls examinee access to the CAT system; and lists the status of examinees in the system. The strategy manager textfile program allows access to data structures used by different strategies of CAT methods; currently, only a Bayesian method is implemented. The graphics management program includes the graphics editor driver and main menu graphics procedures. The CAT diagnostic program is a procedure for detecting problems in the CAT system files. The miscellaneous textfiles program initializes and formats files. (DWH)

****
*   Reproductions supplied by EDRS are the best that can be made   *
*   from the original document.   *
****

TM

ED247242

SUPPLEMENT TO
NPRDC TR 84-33

MARCH 1984

MICROCOMPUTER NETWORK FOR COMPUTERIZED ADAPTIVE TESTING (CAT): PROGRAM LISTING

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☒ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

- Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Navy Personnel
R&D Center
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED

NAVY PERSONNEL RESEARCH
AND
DEVELOPMENT CENTER
San Diego, California 92152

TM 840 393

SUPPLEMENT TO
NPRDC TR 84-33

March 1984

MICROCOMPUTER NETWORK FOR COMPUTERIZED ADAPTIVE TESTING (CAT): PROGRAM LISTING

Baldwin Quan
Thomas A. Park
Gary Sandahl
John H. Wolfe

Reviewed by
James R. McBride

Approved by
Martin F. Wiskoff

Released by
J. W. Renard
Captain, U.S. Navy
Commanding Officer

Navy Personnel Research and Development Center
San Diego, California 92152

CONTENTS

		Page
CATPROJECT.TEXT	CAT system driver textfile	1
ADMIN.DIR	Subdirectory - Test administration textfiles	7
ADMIN.TEXT	Test administration driver	9
A.DEC.TEXT	Declarations for test administration	13
A.1UTL.TEXT	Utilities	21
A.2UTL.TEXT	Utilities	37
A.ESUMM.TEXT	Write Examinee data to textfile end of session	45
A.IO.TEXT	I/O routines	52
A.PROCT.TEXT	Proctor routine	54
A.CF.TEXT	Computer familiarization	59
A.LOGIN.TEXT	Log in of examinee	71
A.GI.TEXT	General instructions	77
A.INITE.TEXT	Initialize examinee subtest record before test	80
A.LOADT.TEXT	Load a subtest to give	82
A.QUEST.TEXT	Select a question to give	89
A.UABIL.TEXT	Update the examinee's ability and variance	98
A.FBACK.TEXT	Give feedback to the examinee	100
A.PASVAB.TEXT	Calculate predicted ASVAB true score	105
PMGR.DIR	Subdirectory - parameter management textfiles	109
P.MGR.TEXT	System parameter set up driver	111
P.UTL.TEXT	Utilities	115
P.VIEW.TEXT	Look at testing parameters	119
P.SP.TEXT	Configures testing parameters	121
P.FBACK.TEXT	Configures feedback parameters	133
TMGR.DIR	Subdirectory - test manager textfiles	139
T.MGR.TEXT	Test manager driver	141
T.1UTL.TEXT	Utilities	146
T.SUBRT.TEXT	Utilities - subroutines	150
T.1SUBRT.TEXT	Utilities - subroutines	161
T.IO.TEXT	I/O routines	171
T.GET1.TEXT	Fetch subtest part 1	173
T.GET2.TEXT	Fetch subtest part 2	181
T.INSTR.TEXT	Manage subtest instructions	193
T.SAMPLES.TEXT	Manage sample questions for subtest	198
T.LIST.TEXT	List subtests in database	207
T.NEW.TEXT	Create a new subtest	209
T.DELETE.TEXT	Delete a subtest	211
T.FLOPPY.TEXT	Write subtest to floppy	212
T.SEARCH.TEXT	Search for duplicate questions	215

Page

EMGR.DIR .... Subdirectory - examinee manager textfiles 223
  E.MGR.TEXT .... Examinee manager driver 225
  E.UTL.TEXT .... Utilities 233
  E.SUBRT.TEXT .... Utilities - subroutines 237
  E.IO.TEXT .... I/O routines 239
  E.LOGIN.TEXT .... Log in examinees, allow access to system 241
  E.FETCH.TEXT .... Get examinee to put in personal data 243
  E.STATUS.TEXT .... Display status of all examinees in system 253
  E.DELETE.TEXT .... Deletes an examinee from database 254
  E.ENDOFDAY.TEXT .... Backup finished examinees 255
  E.SUMMARY.TEXT .... Write examinee data to textfile for SPSS 257
  E.ZERO.TEXT .... Zero out examinee database 269

SMGR.DIR .... Subdirectory - strategy manager textfiles 271
  S.MGR.TEXT .... Strategy manager driver 273
  S.UTL.TEXT .... Utilities 282
  S.NEW.TEXT .... Make new info table 286
  S.VERF.TEXT .... Verify info table 292
  S.MODF.TEXT .... Modify info table 294
  S.LIST.TEXT .... List info table 296
  S.FIND.TEXT .... Queries into info table 300
  S.ANALYZE.TEXT .... Analysis on info table values 305

GMGR.DIR .... Subdirectory - graphics management 313
  G.MGR.TEXT .... Graphics editor driver 315
  G.UTL.TEXT .... Utilities 319
  G.SUBRT.TEXT .... Main menu graphics procedures 323
  G.1SUBRT.TEXT .... Compression routines to krunch graphics 332
  G.2SUBRT.TEXT .... More compression routines. 346

DMGR.DIR .... Subdirectory - CAT diagnostic program 357
  D.MGR.TEXT .... Diagnostic driver 359
  D.UTL.TEXT .... Utilities 362
  D.SEARCH.TEXT .... Searches a subtest for duplicate questions 367
  D.INFOTAB.TEXT .... Checks validity of info tables 381
  D.GRAFIX.TEXT .... Checks to see if graphics files exist 385

MISC.DIR .... Subdirectory - miscellaneous textfiles. 391
  CATFORMAT.TEXT .... Formats new files for system 393
  CATKRUNCH.TEXT .... Will crunch the question text file 403

CATPROJECT.TEXT
CAT System Driver Textfile

```
(**************************************************************************************)
(*                                                                                    *)
(*                   COMPUTER ADAPTIVE TESTING (CAT) PROJECT                          *)
(*                                                                                    *)
(*               Navy Personel Research and Development Center                        *)
(*                             San Diego, California                                  *)
(*      VERSION [ 1.03 ]                                     Feb 28, 1983             *)
(*                                                                                    *)
(**************************************************************************************)
(*                                                                                    *)
(*      Textfile : CATPROJECT.TEXT                  Volume : TFILES                   *)
(*      Codefile : CATPROJECT.CODE                  Volume : CATDATA                  *)
(*                                                                                    *)
(**************************************************************************************)
(*                                                                                    *)
(*      Execution of this program allows access to the major subprograms of           *)
(* the Computer Adaptive Testing System.  These are :                                 *)
(*                                                                                    *)
(*               1. ADMIN   (test administration)                                     *)
(*               2. P.MGR   (test parameters configuration)                           *)
(*               3. T.MGR   (test questions database management)                      *)
(*               4. E.MGR   (examinee database management)                            *)
(*               5. S.MGR   (strategy database management)                            *)
(*               6. G.MGR   (graphics database management)                            *)
(*               7. D.MGR   (system diagnostic program)                               *)
(*                                                                                    *)
(*      The user can execute any of the above subprograms or return to the            *)
(* Pascal operating system from this program.here.  Upon completion of any           *)
(* of the above programs, control is restored to this program by use of the          *)
(* function SETCHAIN.  This allows access to all major codefiles of the CAT          *)
(* Project with one main driver program.                                              *)
(*                                                                                    *)
(**************************************************************************************)

PROGRAM CATPROJECT;
USES CHAINSTUFF;

CONST BELL = 7;
       VERSION = '[1.03]';

TYPE SETOFCHAR = SET OF CHAR;

VAR COMMAND,
    OUTPUT : CHAR;

    (* ............................................................................ *)

    (* clear the screen *)
    PROCEDURE PAGE(DUMMY : CHAR);
    BEGIN
      WRITE(CHR(28));
      GOTOXY(0,0);
    END;   (* page *)

    (* ---------------------------------------------------------------------------- *)

    (**** rings the bell ****)
    PROCEDURE SQUAWK;
    BEGIN
      WRITE(CHR(BELL));
    END;  (* squawk *)

    (* ---------------------------------------------------------------------------- *)

    (* read an acceptable character from the keyboard *)
    FUNCTION GETCHAR(OKSET : SETOFCHAR;
                     FLUSHQUEUE,ECHO,BEEP : BOOLEAN) : CHAR;
    VAR MASK : PACKED ARRAY[0..0] OF CHAR;
    BEGIN
      IF FLUSHQUEUE THEN UNITCLEAR(2); (* flush buffer *)
      REPEAT
        UNITREAD(2,MASK,1);
        IF BEEP AND NOT (MASK[0] IN OKSET) THEN SQUAWK;

      UNTIL MASK[0] IN OKSET;
      IF ECHO AND (MASK[0] IN [CHR(32)..CHR(126)]) THEN
        WRITE(MASK[0]);
      GETCHAR := MASK[0];
    END;  (* getchar *)

    (* ------------------------------------------------------------------------- *)

    (* displays which program you can branch to *)
    PROCEDURE MENU;
    VAR X,Y : INTEGER;
    BEGIN
      PAGE(OUTPUT);
      X := 16;
      Y := 8;
      GOTOXY(21,0);
      WRITE('CAT PROJECT MENU ',VERSION);
      GOTOXY(0,4);
      WRITE('Select one of the following programs by entering its number.');
      GOTOXY(X,Y);
      WRITE('1.  QUIT');
      GOTOXY(X,Y+1);
      WRITE('2.  TEST ADMINISTRATION');
      GOTOXY(X,Y+2);
      WRITE('3.  CONFIGURE TEST PARAMETERS');
      GOTOXY(X,Y+3);
      WRITE('4.  TEST DATABASE MANAGEMENT');
      GOTOXY(X,Y+4);
      WRITE('5.  EXAMINEE DATABASE MANAGEMENT');
      GOTOXY(X,Y+5);
      WRITE('6.  STRATEGY DATABASE MANAGEMENT');
      GOTOXY(X,Y+6);
      WRITE('7.  GRAPHICS EDITOR');
      GOTOXY(X,Y+7);
      WRITE('8.  CAT SYSTEM DIAGNOSTICS');
      GOTOXY(X,Y+10);
      WRITE('Enter Choice # : ');
    END;      (* menu *)

(* ............................................................................. *)

(**** main program ****)
BEGIN

  (* display options *)
  MENU;

  (* get selection *)
  COMMAND := GETCHAR(['1'..'8'],TRUE,TRUE,TRUE);

  (* set the program to branch to *)
  CASE COMMAND OF
     '1' : ;
     '2' : SETCHAIN('CATDATA:ADMIN');
     '3' : SETCHAIN('CATDATA:P.MGR');
     '4' : SETCHAIN('CATDATA:T.MGR');
     '5' : SETCHAIN('CATDATA:E.MGR');
     '6' : SETCHAIN('CATDATA:S.MGR');
     '7' : SETCHAIN('CATDATA:G.MGR');
     '8' : SETCHAIN('CATDATA:D.MGR');
  END;

  IF COMMAND <> '1' THEN
  BEGIN  (*1*)
    PAGE(OUTPUT);
    GOTOXY(15,10);
    WRITE('Loading ');
    CASE COMMAND OF
     '2' : WRITE('Test Administration Program.');
     '3' : WRITE('Test Configuration Program.');
     '4' : WRITE('Test Manager Program.');
```

```
      '5' : WRITE('Examinee Manager Program.');
      '6' : WRITE('Strategy Manager Program.');
      '7' : WRITE('Graphics Manager Program.');
      '8' : WRITE('Cat System Diagnostic Program');
    END; (* cases *)
  END;    (*1*)

END.    (* catproject *)
```

ADMIN.DIR:
Subdirectory - Test Administration Textfiles

```
(*$S++*)
(**************************************************************************)
(*                                                                       *)
(*     Textfile : ADMIN.DIR/ADMIN.TEXT          Volume : TFILES          *)
(*     Codefile : ADMIN.CODE                    Volume : CATDATA         *)
(*                                                                       *)
(**************************************************************************)
(* File Last Modified  :   October 7 1983                NPROC          *)
(**************************************************************************)
(*                              VERSION 1.05                             *)
(**************************************************************************)
(*      1. 256 K of memory required.                                     *)
(*      2. Graphics questions capability                                 *)
(*      3. Thunder clock required.                                       *)
(**************************************************************************)
(*                                                                       *)
(* Description :                                                         *)
(*      This program gives examinees a prescribed set of subtests.  It begins *)
(* with an introduction to the computer, then logs in an examinee, and gives *)
(* him the subtests.  The main modules of the program are :              *)
(*        1. Proctor routines.                                           *)
(*        2. General instructions   (computer familiarization)           *)
(*        3. Login procedures.                                           *)
(*        4. Examinee pretest initialization.                            *)
(*        5. Loading subtest directory and appropriate strategy data structure *)
(*           (if any).                                                   *)
(*        6. Administering a question.                                   *)
(*        7. Updating the examinee ability level.                        *)
(*        8. Feedback routines.                                          *)
(*        9. Examinee backup to compressed file for data analysis.       *)
(*                                                                       *)
(*      This program basically controls the flow of execution through these *)
(* modules.  All communication between modules occurs with global variables *)
(* in the program.  Three major loops exist :                            *)
(*      (1) Giving continuous sessions.                                  *)
(*      (2) Giving a sequence of subtests.                               *)
(*      (3) Giving a sequence of questions.                              *)
(* Control through these loops and modules can be altered by commands from *)
(* the proctor routines.                                                 *)
(*                                                                       *)
(**************************************************************************)

PROGRAM TESTADMINISTRATION;

USES PGRAF,
     REALMODES,
     TRANSCEND,        (* needed for real functions *)
     APPLESTUFF,       (* needed for randomizer *)
     CHAINSTUFF;       (* allows return to project driver *)

(* const,type,& var declarations *)
   (*$I /TFILES/ADMIN.DIR/A.DEC.TEXT *)

(* proctor routines *)
   (*$I /TFILES/ADMIN.DIR/A.PROCT.TEXT *)

(* computer familiarization    *)
   (*$I /TFILES/ADMIN.DIR/A.CF.TEXT *)

(* general instructions part 2 *)
   (*$I /TFILES/ADMIN.DIR/A.GI.TEXT *)

(* short version - doesn't get personal data *)
   (*$I /TFILES/ADMIN.DIR/A.LOGIN.TEXT *)

(* initialize examinee test rscores *)
   (*$I /TFILES/ADMIN.DIR/A.INITE.TEXT *)

(* subtest load *)
   (*$I /TFILES/ADMIN.DIR/A.LOADT.TEXT *)

(* feedback routines *)
```

```
   (*$I /TFILES/ADMIN.DIR/A.FBACK.TEXT *)

(* writes examinee summary to text file *)
   (*$I /TFILES/ADMIN.DIR/A.ESUMM.TEXT *)

(* disk io *)
   (*$I /TFILES/ADMIN.DIR/A.IO.TEXT *)

(* utility subroutines *)
   (*$I /TFILES/ADMIN.DIR/A.1UTL.TEXT *)

(* utility subroutines which can call proctor *)
   (*$I /TFILES/ADMIN.DIR/A.2UTL.TEXT *)

(* ability update method *)
   (*$I /TFILES/ADMIN.DIR/A.UABIL.TEXT *)

(* administer question/get response *)
   (*$I /TFILES/ADMIN.DIR/A.QUEST.TEXT *)

(* calculate predicted asvab score for paper/pencil *)
   (*$I /TFILES/ADMIN.DIR/A.PASVAB.TEXT *)

(* --------------------------------------------------------------------------*)

BEGIN  (* main program *)

  (* initialize variables *)
  (* File : CAT.1UTL *)
  INITCAT;

  (* give continuous sessions *)
  REPEAT

    (* display a session header and initialize *)
    (* File : CAT.1UTL *)
    SESSIONHEADER;

    (* give computer familarization *)
    (* File : CAT.CF *)
    IF NOT SKIPFAM THEN
    COMPUTERFAMILARIZATION;

    (* give general instructions *)
    (* File : CAT.GI *)
    IF (FLOWCODE = CONTINUE) AND (NOT SKIP) THEN
      GENERALINSTRUCTIONS;

    (* mark time spent in orientation *)
    (* TIME is in File : CAT.1UTL *)
    OTIME := TRUNC(TIME - SESSSTART);

    IF FLOWCODE = CONTINUE THEN
    BEGIN  (*1*)

      (* log-in examinee *)
      (* File : CAT.LOGIN *)
      LOGIN;

      (* controls initialization of test scores *)
      INITSCORES := FALSE;

      (* give a loop of subtests *)
      WHILE ((FLOWCODE = CONTINUE) OR (FLOWCODE = NEXTST))  (* normal flow *)
            AND (TINDEX < GMAXSUBTEST) DO (* max # of tests not taken *)
      BEGIN  (*2*)
        FLOWCODE := CONTINUE;

        (* index into setup parameter arrays, also saved as *)
        (* 'examinee.lasttest' for abnormal session exits   *)
        TINDEX := TINDEX + 1;

        (* mark where examinee currently is in testing sequence *)
```

```
EXAMINEE.LASTTEST := TINDEX;

(* initialize the examinee, test/parameters *)
(* File : CAT.INITE *)
INITEXAMINEE;

(* give a test if anymore to give , if currtest is >= 0 *)
(* that means there is a test to be given.             *)
IF CURRTEST <= MAXSUBTESTS THEN
BEGIN  (*3*)

  (*  clock the beginning of the subtest *)
  (* File : CAT.1UTL *)
  SUBTSTART := TIME;

  (* load the subtest according to strategy, give test *)
  (* instructions and sample questions.                *)
  (* File : CAT.LOADT *)
  LOADTEST;

  (* mark the time spent in instructions and samples *)
  TESTS.STINSTRTIME := TRUNC(TIME - SUBTSTART);

  (* give a loop of questions *)
  WHILE (FLOWCODE = CONTINUE)               (* normal flow mode *)
        AND (QNUM < CURRTLENGTH) DO (* max # of questions not taken *)
  BEGIN  (*4*)

    (* select question according to strategy, display , *)
    (* get response, save response, & update counters.  *)
    (* File : CAT.QUEST *)
    ADMINISTERQUESTION;

    (* if question was answered *)
    IF (FLOWCODE = 1) OR (FLOWCODE = 6) THEN
    BEGIN  (*5*)

      (* save the ability and variance *)
      TESTS.ITEMINFO[QNUM].THETA := CURRABILITY;
      TESTS.ITEMINFO[QNUM].ERROR := CURRVARIANCE;

      (* give feedback if any *)
      (* File : CAT.FBACK *)
      FEEDBACK(FBITEM,SPARAMS.ITEMFB,SPARAMS.ITEMOUTPUT);

      (* increment array index for next question *)
      QNUM := QNUM + 1;

      IF CURRVARIANCE <= S_PARAMS.CX_ERROR[T_INDEX] THEN
        FLOWCODE := NEXTST;

    END;   (*5*)
  END;  (*4*)   (* loop of questions *)
  PAGE(OUTPUT);

  (* save final ability/variance *)
  TESTS.ESTABILITY := CURRABILITY;
  TESTS.VARIANCE := CURRVARIANCE;


  (* save amount of time spent at this subtest *)
  TESTS.STTIME := TRUNC(TIME - SUBTSTART);


  (* subtest feedback *)
  (* File : CAT.FBACK *)
  FEEDBACK(FBSUBTEST,SPARAMS.SUBTESTFB,SPARAMS.SUBTESTOUTPUT);


  (* look-up and save predicted asvab *)
  (* File : A.PASVAB   *)

   EXAMINEE.PREDASVAB[TINDEX]:=PREASVABCALC(CURRABILITY);
```

13

```
          (* check subtest stop flag *)
          (* IF FLOWCODE = 1     THEN
              CALLPROCTOR(5,FLOWCODE); *)


          (* write subtest results to disk *)
          (* File : CAT.IO *)
          UPDATERESULTS(TNUM);

          (* is taking suceeding subtests from the beginning and must      *)
          (* initialize his test scores.                                   *)
          INITSCORES := TRUE;


          (* save current examinee data *)
          (* File : CAT.IO *)
          UPDATEEXAMINEE(ERECNUM);

        END;  (*3*)

        (* increment file record # to store next subtest *)
        TNUM := TNUM + 1;
      END;  (*2*)   (* loop of subtests  *)

      (* give session feedback *)
      IF FLOWCODE = 1 THEN
        EXAMINEE.LASTTEST := 128;  (* flag examinee totally done *)

      (* if examinee leaves session in middle of timed test, save time  *)
      (* expended.                                                      *)
      IF (FLOWCODE = 6) AND (CURRSTRAT = TIMED) THEN
        EXAMINEE.PREVTIMELASTTEST := ELASPEDTIME;

    END;  (*1*)  (* continue after general instructions *)
    IF FLOWCODE = 1 THEN
      FEEDBACK(FBSESSION,SPARAMS.SESSIONFB,SPARAMS.SESSIONOUTPUT);

    (* give closing session message and save some data *)
    (* File : CAT.IUTL *)
    CLOSING_MESSAGE;

    (* save examinee data in coded textfile if done *)
    (* File : CAT.ESUMM *)
    IF (EXAMINEE.LASTTEST > GMAXSUBTEST) AND (NOT DEMOFLAG) THEN
    BEGIN
      CALLPROCTOR(5,FLOWCODE);  (* last proctor call *)
      EXAMINEE.NUMPROC:=EXAMINEE.NUMPROC -1;
      ESUMMARY;
    END;

  UNTIL FLOWCODE = ENDOFDAY;

END.  (* catdemo       *)
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : ADMIN.DIR/A.DEC.TEXT          Volume : TFILES              *)
(*      Codefile : ADMIN.CODE ('Include' file)  Volume : CATDATA             *)
(*                                                                            *)
(******************************************************************************)
(*  File Last Modified  :     October 7 1983                     NPROC            *)
(******************************************************************************)

(* constant, var, & type declarations *)

CONST (* ascii values *)
      BELL        = 7;
      LARROW      = 8;
      RARROW      = 21;
      RET         = 13;
      ESC         = 27;
      SPACE       = 32;
      ASCIIOFFSET = 48; (* ascii zero *)
      NIL         = -1; (* signifies not used/bad value *)

      (* these are ascii for the a b c d e keys *)
      AKEY = 74;     (* ascii J *)
      BKEY = 75;     (* ascii K *)
      CKEY = 76;     (* ascii L *)
      DKEY = 59;     (* ascii ; *)
      EKEY = 39;     (* ascii ' *)



      (* flowcode values, serves as path gates for program flow  *)
      CONTINUE = 1;
      EXITGI    = 2;
      EXITLOGIN= 3;
      EXITSQ    = 4;
      NEXTST    = 5;
      NEXTSESS = 6;
      ENDOFDAY = 7;


      (* flowcode values for ghost routines *)
      OLD = 1;
      SINGLE = 2;
      AUTO = 3;



      (* keypad assignments *)
      HELPKEY      = 92; (* ascii \ *)
      YESKEY       = 13; (* ascii <enter> *)
      NOKEY        = 45; (* ascii - *)
      ERASEKEY     = 46; (* ascii . *)

      (* maximum size of ascii buffer *)
      MAXITEMBUF = 2047;

      (* line character buffer size, 0 - 79 = 80 char *)
      MAXLINEBUF  = 79;

      (* question textfile control codes *)
      GOTOFLAG     = 128;  (* signals gotoxy condition *)
      PAGEFLAG     = 129;  (* signals another page of text *)
      UNUSEDFLAG   = 130;  (* signals byte not used *)
      ENDITEM      = 131;  (* signals end of text for question *)

      (* these files must reside on disk ! *)
      DATANAME  = 'CATDATA:ITEMPOOL.DATA';      (* question data *)
      TEXTNAME  = 'QTEXT:ITEMTEXT.DATA';        (* question text *)
      INDEXNAME = 'CATDATA:TESTINDEX.DATA';     (* test directory *)
      EINDEX    = 'CATDATA:EINDEX.DATA';        (* examinee directory *)
      INFONAME  = 'CATDATA:EINFO.DATA';         (* examinee data *)
      RESULTS   = 'CATDATA:ERESULTS.DATA';      (* examinee test scores *)
      TABNAME    = 'CATDATA:TABINFO.DATA';       (* information tables *)
      SETUPDATA  = 'CATDATA:PARAMETERS.DATA';    (* test default parameters *)


      (* slots available in test directory,  0 - 20 *)
      MAXSUBTESTS = 20;

      (* maximum question pool per test, 0 - 300 *)
      MAXITEMPOOL = 300;

      (* maximum # of sample questions you can have *)
      MAXSAMPLES = 5;

      (* slots available in examinee directory, 0 - 50 *)
      MAXEXAMINEE = 50;

      (* max # of questions you can give per test *)
      QUESTIONS    = 20;

      (* max # of subtests you can give *)
      GMAXSUBTEST = 20;

      (* max ability for ghost loop *)
      MAXGTHETA = 3;

      (* min ability for ghost loop *)
      MINGTHETA = -3;

      (* cat strategy codes *)
      NONE = 0;
      B102222 = 1;  (* original baysian *)
      B54321 = 2;
      B108642 = 3;
      TIMED = 4;


      (* information table dimensions , used for bayesian strategy *)
      (* 36 rows and 20 columns *)
      INFOROW     = 36;
      INFOCOLUMN = 20;

      (* demo id # , no files are written to disk with this id # *)
      DEMOID = 'DEMO     ';

      (* printer unit number *)
      UNITNUMPRINTER = 'PRINTER:';

      (* code number to flag compressed graphics *)
      COMPRESSED = 1.0;

      VERSION = '[1.05]';


TYPE (* different types of ways to answer a question *)
     ITEMRESPONSES = (CHARVALUE,      (* normal multiple choice *)
                      INTVALUE,       (* integer value as answer *)
                      SEVENCHR);      (* seven characters saved as answer *)


     (* ability loop range *)
     LOOPRANGE = MINGTHETA..MAXGTHETA;


     (* feedback levels *)
     TYPEFEEDBACK = (FBITEM, FBSUBTEST, FBSESSION);

     (* type of question response *)
     SEVENTYPE = PACKED ARRAY[1..7] OF CHAR;

     (* social security number *)
     IDTYPE = PACKED ARRAY[0..8] OF CHAR;

     (* all characters *)
```

```
SETOFCHAR = SET OF CHAR;

(* information table *)
TABLE = ARRAY[1..INFOCOLUMN,1..INFOROW] OF INTEGER;

(* test directory *)
DIRDATA = PACKED RECORD

                   (* true ==>  this record not occupied *)
                   UNUSED     : BOOLEAN;

                   (* name of subtest *)
                   TESTNAME   : STRING;

                   (* directory of pool of questions in subtest *)
                   ITEMCODE   : PACKED ARRAY
                                  [0..MAXITEMPOOL]
                                  OF INTEGER;  (* question code # *)
                 END;

(* question ptrs/data , information for each question *)
ITEMDATA = PACKED RECORD

                    (* flags if graphics item *)
                    GRAPHICS : BOOLEAN;

                    (* valid response ranges for multiple choice *)
                    LOWANSWER,
                    HIGHANSWER : CHAR;

                    (* block # in file where text starts *)
                    ITEMBLOCK,

                    (* byte # in block where text starts *)
                    ITEMPTR,

                    (* # of answers if multiple question screen *)
                    ANSWERCOUNT   : INTEGER;

                    (* information parameters for bayesian strategy *)
                    A,B,C,

                    (* currently unused *)
                    PROPCORRECT,
                    POINTBISERIAL,
                    YOPT,
                    XOPT,
                    DUMMY1,
                    DUMMY2,
                    DUMMY3 : REAL;

                    (* correct answer to question *)
                    CASE ATYPE : ITEMRESPONSES OF
                      CHARVALUE : (ANSWER : CHAR);
                      INTVALUE  : (INTANSWER : INTEGER);
                      SEVENCHR  : (CHRANSWER : SEVENTYPE);
                  END;

(* examinee directory *)
INDEX = PACKED ARRAY[0..MAXEXAMINEE] OF PACKED RECORD

                                          (* true ==> trecord available *)
                                          UNUSED : BOOLEAN;

                                          (* social security/id key *)
                                          ID : IDTYPE;
                                        END;

(* examinee personal data *)
EXAMEINFO = PACKED RECORD
```

```
              (* social security # *)
              ID            : IDTYPE;

              (* time spent in computer orientation *)
              ORIENTATIONTIME,

              (* if timed test, last time when session interrupted *)
              PREVTIMELASTTEST,

              (* number of proctor calls *)
              NUMPROC,

              (* total time spent at computer *)
              TOTTIMECONSOLE,

              (* number of key in errors *)
              NUMERRORS,

              (* # of last test taken - 1, eg.- if on test 5 then *)
              (* this variable holds a value of 4.                *)
              LASTTEST      : INTEGER;

              (* date of log in *)
              DATE          : PACKED ARRAY[0..5] OF CHAR;

              (* time at log in *)
              TIME          : PACKED ARRAY[0..3] OF CHAR;

              (* testing configuration given to this examinee *)

              (* record # of subtest directories *)
              TESTORDER,

              (* adaptive strategy *)
              STRATEGY,

              (* # of questions per test *)
              TESTLENGTH    : PACKED ARRAY[1..GMAXSUBTEST]
                              OF 0..128;

              (* predicted asvab scores for paper/pencil *)
              PREDASVAB,

              (* initial variance for each test *)
              CKERROR       : ARRAY[1..GMAXSUBTEST] OF REAL;

              (* flags to control flow of subtests *)
              SUBSTOP       : PACKED ARRAY[1..GMAXSUBTEST] OF BOOLEAN;

            END;

(* question scores , data on examinee with respect to question *)
ITEM = PACKED RECORD

         (* true ==> answered correctly, for seven char answer *)
         ACORRECT : PACKED ARRAY[0..6] OF BOOLEAN;

         (* number of answers *)
         ACOUNT,

         (* which question he took, code # *)
         ITEMNUM  : INTEGER;

         (* true ==> answered it correctly *)
         CORRECT  : BOOLEAN;

         (* ability after answering this question *)
         THETA,

         (* variance after answering this question *)
         ERROR,
         (* time spent answering this question *)
         LATENCY  : REAL;

         (* how he answered the question *)
         CASE RTYPE : ITEMRESPONSES OF
           CHARVALUE : (RESPONSE : CHAR);
           INTVALUE  : (INTRESPONSE : INTEGER);
           SEVENCHR  : (CHRRESPONSE : SEVENTYPE);
       END;

(* test scores of examinee results  *)
SUBTEST = PACKED RECORD

            (* time taken for subtest *)
            STTIME,

            (* time taken for subtest instructions *)
            STINSTRTIME,

            (* number of proctor calls during subtest *)
            STPROCTCALLS,

            (* modified time - time for subtest minus overhead *)
            MODSTTIME : INTEGER;

            (* number of questions he answered *)
            NUMITEMS,

            (* # of correct responses *)
            NUMCORR : 0..128;

            (* final estimate of ability *)
            ESTABILITY,

            (* final variance of ability *)
            VARIANCE    : REAL;

            (* results on question level *)
            ITEMINFO    : PACKED ARRAY[0..QUESTIONS] OF ITEM;
          END;

(* set-up parameters *)
SETUPINFO = PACKED RECORD

              (* flags # tests in sequence, > 20 = no subtest *)
              SUBORDER,

              (* strategy setup *)
              SUBSTRAT        : PACKED ARRAY[1..GMAXSUBTEST]
                                  OF 0..128;

              (* question feedback code *)
              ITEMFB,

              (* question feedback output code *)
              ITEMOUTPUT,

              (* subtest feedback code *)
              SUBTESTFB,

              (* subtest feedback destination code *)
              SUBTESTOUTPUT,

              (* session feedback code *)
              SESSIONFB,

              (* session feedback destination; screen/printer *)
              SESSIONOUTPUT  : 0..128;

              (* subtest stop flag *)
              SUBSTOP         : PACKED ARRAY[1..GMAXSUBTEST] OF BOOLEAN;
```

```
               (* subtest length *)
               SUBLENGTH        : PACKED ARRAY[1..GMAXSUBTEST]
                                  OF 0..128;

               (* initial variance *)
               CKERROR          : ARRAY[1..GMAXSUBTEST] OF REAL;

             END;


VAR (* 0 through 9 *)
    DIGITS,

    (* helpkey,yeskey,nokey,erasekey *)
    CONTROLKEYS,

    (* digits + control keys *)
    KEYPAD : SETOFCHAR;

    (* ability for ghost routines *)
    ABLOOP: LOOPRANGE;


    (* string character buffer *)
    LINEBUF : PACKED ARRAY[0..MAXLINEBUF] OF CHAR;
    (* contains row and column index in the infotable *)
    SAVERC : ARRAY[1..4] OF INTEGER;

    LEVEL,                        (* row location index for the stay routine *)
    OTIME,                        (* time spent in computer orientation *)
    ELASPEDTIME,                  (* elasped time in seconds for timed tests *)
    MAXTIME,                      (* maximum time in seconds for timed tests *)
    RITEPREFETCH,                 (* directory slot of prefetch if correct *)
    WRONGPREFETCH,                (* directory slot of prefetch if wrong *)
    QPREFETCH,                    (* next question to be selected, = one of above *)
    FLOWCODE,                     (* controls flow in driver  *)
    GHOSTFLOW,                    (* flow control for ghost routines *)
    CURRSTRAT,                    (* current test strategy *)
    CURRTLENGTH,                  (* current # of questions to give *)
    CURRTEST,                     (* current record # of subtest *)
    TINDEX,                       (* index into test order array *)
    ERECNUM,                      (* current examinee record # *)
    QNUM,                         (* current array index for question results *)
    TNUM : INTEGER;               (* current test result record file number *)

    SESSSTART,                 (* time at start of session *)
    SUBTSTART,                 (* time at start of subtest *)
    GTHETA,                    (* ghost theta *)
    RITEABILITY,               (* precalculated ability if answered correctly *)
    WRONGABILITY,              (* precalculated ability if answered wrong *)
    RITEVARIANCE,              (* precalculated variance if correct *)
    WRONGVARIANCE,             (* precalculated variance if wrong *)
    CURRABILITY,               (* current examinee ability level *)
    CURRVARIANCE : REAL;       (* current variance of ability *)

    STAY,                     (* true ==> program runs at one ability level *)
    GHOST,                      (* true ==> program in self test mode *)
    REVERSEVIDEO,               (*  true ==> screen is inverse  *)
    FORTYCOLUMN,                (*  true ==> screen is 40 columns *)
    GRAFIX,                     (* true ==> console is currently in graphics *)
    SKIP,                       (* true ==> skip to log in *)
    SKIPFAM,                    (* true ==> skip computerfamiliarization *)
    SAMPLEQUESTION,             (* true ==> giving sample question *)
    TIMEOUT,                    (* true ==> time up for subtest *)
    FIRSTQUESTION,              (* true ==> first question given, don't prefetch *)
    DEMOFLAG,                   (* true ==> don't update files *)
    INITSCORES,                 (* true ==> init test record scores *)
    CURRSUBSTOP,                (* true ==> halt session after subtest *)
    RIGHT,                      (* TRUE ==> examinee answered question correct *)
    NEWEXAMINEE : BOOLEAN;      (* true ==> first time examinee has logged in *)
    OUTPUT : char;           (* dummy variable for page function *)

    SYSTEMDATE : PACKED ARRAY[0..5] OF CHAR;

    (* buffer which serves to hold item questiun ascii or compressed *)
    (* graphics information .                                        *)
    TRIX : RECORD CASE INTEGER OF
            1 : (ITEMBUF : ARRAY[0..1023] OF INTEGER);
            2 : (ASCIIBUF : PACKED ARRAY[0..2047] OF 0..139);
          END;


    (* test directory *)
    DIRECTORY : DIRDATA;
    FILEDIRECTORY : FILE OF DIRDATA;


    (* test question ptrs/data *)
    ITEMINFO : ITEMDATA;
    FILEITEMINFO : FILE OF ITEMDATA;


    (* file of ascii codes, control #'s *)
    ITEMTEXT : FILE;


    (* examinee personal data *)
    EXAMINEE : EXAMEINFO;
    FILEEXAMINEE : FILE OF EXAMEINFO;


    (* examinee directory *)
    DIR : INDEX;
    EDIR : FILE OF INDEX;


    (* examinee test results *)
    TESTS : SUBTEST;
    FILETESTS : FILE OF SUBTEST;


    (*** set-up variables ***)
    SPARAMS : SETUPINFO;
    FILESPARAMS : FILE OF SETUPINFO;


    (* info table *)
    INFOTABLE : TABLE;
    INFOFILE  : FILE OF TABLE;

    (* store used questions here *)
    USEDQ : PACKED ARRAY[0..QUESTIONS] OF INTEGER;

    (* for text file listings *)
    DEST : TEXT;

    (* debugging flag *)
    trace : boolean;

PROCEDURE LOADINDEX;  FORWARD;
PROCEDURE LOADEXAMINEE(RECNUM : INTEGER);  FORWARD;
PROCEDURE UPDATEEXAMINEE(RECNUM : INTEGER);  FORWARD;
PROCEDURE UPDATEINDEX;  FORWARD;
PROCEDURE UPDATERESULTS(RECNUM : INTEGER); FORWARD;
PROCEDURE LOADRESULTS(RECNUM : INTEGER); FORWARD;
PROCEDURE LOADINFO(RECNUM : INTEGER); FORWARD;
PROCEDURE LOADPARAMS; FORWARD;

FUNCTION GHOSTDIGIT : CHAR;  FORWARD;
FUNCTION GHOSTKEY : CHAR;    FORWARD;
PROCEDURE PAGE(DUMMY : CHAR);  FORWARD;
PROCEDURE SQUAWK;  FORWARD;
FUNCTION GETCHAR(OKSET : SETOFCHAR;
                  FLUSHQUEUE,
                  ECHO,
                  BEEP : BOOLEAN) : CHAR; FORWARD;
PROCEDURE FILLBUF(CHARCNT : INTEGER;
                   OKSET : SETOFCHAR; ERASE : BOOLEAN);  FORWARD;
PROCEDURE SCRCONTROL(I,J,K:INTEGER);  FORWARD;
PROCEDURE TEXT40MODE;  FORWARD;
PROCEDURE TEXT80MODE;  FORWARD;
PROCEDURE INVERSE;     FORWARD;
PROCEDURE NORMALSCR;      FORWARD;
FUNCTION TIME : REAL;  FORWARD;
PROCEDURE GWRITESTR(GSTR : STRING);  FORWARD;
PROCEDURE GGOTOXY(X,Y : INTEGER); FORWARD;
PROCEDURE INITFORGRAFIX;   FORWARD;
PROCEDURE DELAY(SECONDS : INTEGER); FORWARD;
PROCEDURE SESSIONHEADER;  FORWARD;
PROCEDURE CLOSINGMESSAGE; FORWARD;
PROCEDURE STALL;  FORWARD;
PROCEDURE DECODEPRINT(BLOCKNUM, BLOCKPTR : INTEGER); FORWARD;
PROCEDURE BLANKLINES(START,COUNT,ENDCURSOR : INTEGER);  FORWARD;
FUNCTION SLOT(CODE : INTEGER) : INTEGER;  FORWARD;
FUNCTION HASH(KEY : INTEGER) : INTEGER;  FORWARD;
FUNCTION GETINTSTR : INTEGER;  FORWARD;
PROCEDURE GWRITECHR(GCHR : CHAR);  FORWARD;
PROCEDURE GWRITEINT(GINT : INTEGER);  FORWARD;
PROCEDURE GWRITELN;  FORWARD;
PROCEDURE GSTALL;  FORWARD;
FUNCTION GGETCHAR(OKSET : SETOFCHAR;
                  FLUSHQUE,
                  ECHO,
                  BEEP : BOOLEAN) : CHAR;  FORWARD;
PROCEDURE GBLANKLINES(STARTBLANK,
                      BLANKTHISMANY,
                      LEAVECURSOR : INTEGER); FORWARD;
PROCEDURE GFILLBUF(CHARCNT : INTEGER;
                   OKSET : SETOFCHAR; ERASE : BOOLEAN); FORWARD;
PROCEDURE GDECODEPRINT(SUBTESTNUM,
                       ITEMCODE : INTEGER);  FORWARD;
FUNCTION GGETINTSTR : INTEGER;    FORWARD;

PROCEDURE PFILLBUF(CHARCNT : INTEGER;
                   OKSET : SETOFCHAR; ERASE : BOOLEAN;
                   FILLFROM : INTEGER; VAR FFLOW : INTEGER);  FORWARD;
PROCEDURE PSTALL(STALLFROM : INTEGER; VAR SFLOW : INTEGER);  FORWARD;
FUNCTION GETCHRANSWER(CALLEDFROM : INTEGER;
                      VAR PCODE : INTEGER) : CHAR;  FORWARD;
FUNCTION GETINTANSWER(CALLEDFROM : INTEGER;
                      VAR PCODE : INTEGER)  : INTEGER;  FORWARD;
PROCEDURE GETSEVENANSWERS(QSTNUM,QSTCNT,CALLEDFROM,
                                 GBLOCK,GPTR : INTEGER;
                                 VAR A7:SEVENTYPE; VAR PCODE,QACNT : INTEGER);
                          FORWARD;
PROCEDURE GPFILLBUF(CHARCNT : INTEGER;
                   OKSET : SETOFCHAR; ERASE : BOOLEAN;
                   FILLFROM : INTEGER; VAR FFLOW : INTEGER);  FORWARD;
FUNCTION GGETCHRANSWER(CALLEDFROM : INTEGER;
                      VAR PCODE : INTEGER) : CHAR;   FORWARD;
FUNCTION GGETINTANSWER(CALLEDFROM : INTEGER;
                      VAR PCODE : INTEGER)  : INTEGER;  FORWARD;
PROCEDURE GGETSEVENANSWERS(QSTNUM,QSTCNT,CALLEDFROM,
                                 GTEST,GITEM : INTEGER;
                                 VAR A7:SEVENTYPE;
                                 VAR PCODE,QACNT : INTEGER);  FORWARD;
```

```
(*****************************************************************************)
(*                                                                           *)
(*      Textfile : ADMIN.DIR/A.1UTL.TEXT          Volume : TFILES            *)
(*      Codefile : ADMIN.CODE ('Include' f''s)    Volume : CATDATA           *)
(*                                                                           *)
(*****************************************************************************)
(*  File Last Modified  :  October 7 1983                  NPROC             *)
(*****************************************************************************)


FUNCTION GHOSTDIGIT;
BEGIN
  GHOSTDIGIT := CHR((RANDOM MOD 10) + 48);
END;     (* ghost digit *)


FUNCTION GHOSTKEY;
VAR T : INTEGER;
    GKEY : CHAR;
BEGIN
  REPEAT
    T := RANDOM;
    T := (T MOD 5) + 1;
    CASE T OF
      1 : GKEY := 'A';
      2 : GKEY := 'B';
      3 : GKEY := 'C';
      4 : GKEY := 'D';
      5 : GKEY := 'E';
    END;
  UNTIL GKEY IN [ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER];
  GHOSTKEY := GKEY;
END;  (* ghostkey *)



FUNCTION GRANDOM:REAL;
  BEGIN
    GRANDOM:=RANDOM/MAXINT;
  END;


FUNCTION PROB(THETA,A,B,C:REAL):REAL;
  BEGIN
    PROB:= C + (1-C) / (1 + EXP(-1.7 * A  * (THETA-B)));
  END;

FUNCTION NEWGHOSTKEY(THETA:REAL):CHAR;
  VAR
  R,P:REAL;
  ANS:INTEGER;

  BEGIN
    R:=GRANDOM;
    P:=PROB(THETA,ITEMINFO.A,ITEMINFO.B,ITEMINFO.C);
    IF R > P THEN
      BEGIN
        ANS:= ORD(ITEMINFO.ANSWER);
        IF ANS < ORD(ITEMINFO.HIGHANSWER)
        THEN ANS:= SUCC(ANS)
        ELSE ANS:=PRED(ANS);
        NEWGHOSTKEY:= CHR(ANS);
      END
      ELSE
        NEWGHOSTKEY:=ITEMINFO.ANSWER;
   END;


(* clears screen and puts cursor at 0,0 *)
PROCEDURE PAGE;
BEGIN
  WRITE(CHR(28));
  GOTOXY(0,0);
END;    (* page *)


(**** rings the bell ****)
PROCEDURE SQUAWK;
BEGIN
  WRITE(CHR(BELL));
END;  (* squawk *)



(* read an acceptable character from the keyboard *)
FUNCTION GETCHAR;
VAR MASK : PACKED ARRAY[0..0] OF CHAR;
    ASCII : INTEGER;
BEGIN
  IF FLUSHQUEUE THEN UNITCLEAR(2);  (* flush buffer *)
  REPEAT
    UNITREAD(2,MASK,1);
    ASCII := ORD(MASK[0]);
    CASE ASCII OF
      AKEY : MASK[0] := 'A';
      BKEY : MASK[0] := 'B';
      CKEY : MASK[0] := 'C';
      DKEY : MASK[0] := 'D';
      EKEY : MASK[0] := 'E';
    END;
    IF BEEP AND NOT (MASK[0] IN OKSET) THEN SQUAWK;
  UNTIL MASK[0] IN OKSET;
  IF ECHO AND (MASK[0] IN [CHR(32)..CHR(126)])
     AND (NOT (MASK[0] IN CONTROLKEYS)) THEN
    WRITE(MASK[0]);
  GETCHAR := MASK[0];
END;  (* getchar *)



(* read in a string and save in a temporary buffer  *)
PROCEDURE FILLBUF;
VAR I : INTEGER;
    IDCHAR : CHAR;
BEGIN
  I := 0;    (* initialize count of characters *)
  REPEAT
    IF I > (CHARCNT-1) THEN
      (* maximum char typed in/allow only backspace or return *)
      IDCHAR :=
        GETCHAR([CHR(ERASEKEY),CHR(YESKEY)],TRUE,FALSE,TRUE)
    ELSE
    BEGIN
      (* get a character *)
      IDCHAR :=
        GETCHAR(OKSET + [CHR(YESKEY),
                CHR(ERASEKEY)],
                TRUE,TRUE,TRUE);
      IF (IDCHAR IN OKSET) AND (IDCHAR <> CHR(YESKEY)) THEN
      (* save visible character *)
      BEGIN
        LINEBUF[I] := IDCHAR;
        I := I + 1;
      END;
    END;
    IF IDCHAR = CHR(ERASEKEY) THEN
    (* back space key hit *)
    BEGIN
      IF I = 0 THEN    (* no character to backspace over *)
        SQUAWK
      ELSE
      BEGIN
        WRITE(CHR(LARROW));  (* move cursor back one *)
        I := I - 1;          (* adjust string buffer location *)
        IF ERASE THEN
        (* blank out backspaced character *)
        BEGIN
          WRITE(' ');
          WRITE(CHR(LARROW));
          LINEBUF[I] := ' ';
        END;
      END;
    END;
  UNTIL IDCHAR = CHR(YESKEY);
END;  (* fillbuf *)


(* send control characters to screen *)
PROCEDURE SCRCONTROL;
                                        { PASCAL interface to Screen Control}
VAR N: INTEGER;                         { APPLE III Standard Device Drivers}
    G_ARRAY:PACKED ARRAY[0.. 3] OF 0..255;       {....... Pages 34 to 46.}
BEGIN
  G_ARRAY[0]:= I; G_ARRAY[1]:=J; G_ARRAY[2]:=K;
  UNITWRITE(1,G_ARRAY,3,,12);
END;    (* scrcontrol *)


(* switch to 40 column screen *)
PROCEDURE TEXT40MODE;
BEGIN
  SCRCONTROL(16,0,28);  {Text mode 40BW, followed by clearscreen.}
END;  (* text40mode *)


(* switch to 80 column screen *)
PROCEDURE TEXT80MODE;
BEGIN
  SCRCONTROL(04,0,0);  {Restore Viewport to its original condition.}
  SCRCONTROL(16,2,28);{ Text Mode 80, followed by clearscreen.}
END;    (* text80mode *)


(* turn on reverse video *)
PROCEDURE INVERSE;
BEGIN
  SCRCONTROL(18,0,0);
END;     (* inverse *)


(* restore normal console *)
PROCEDURE NORMALSCR;
BEGIN
  SCRCONTROL(17,0,0);
END;     (* normal *)


(* returns the # seconds elasped since start of the day *)
FUNCTION TIME;
TYPE TIMERECORD = RECORD
                    S : PACKED ARRAY[1..12] OF CHAR;
                  END;
VAR HOUR,
    MINUTE,
    SECOND : REAL;

    T : TIMERECORD;
    TFILE : FILE OF TIMERECORD;

BEGIN

  RESET(TFILE,'.CLOCK');
  T := TFILE^;
```

25

```
  HOUR := ((ORD(T.S[6]) - 48.0) * 10.0) +
          (ORD(T.S[7]) - 48.0);
  MINUTE := ((ORD(T.S[8]) - 48.0) * 10.0) +
          (ORD(T.S[9]) - 48.0);
  SECOND := ((ORD(T.S[10]) - 48.0) * 10.0) +
          (ORD(T.S[11]) - 48.0);
  TIME := (HOUR * 3600.0) + (MINUTE * 60.0) + SECOND;

END;    (* time *)


(* does a write to graphics screen for string values *)
PROCEDURE GWRITESTR;
BEGIN
  UNITWRITE(3,GSTR[1],LENGTH(GSTR),0,12);
END;  (* gwritestr *)


(* do a gotoxy to the graphics screen, treats graphics screen as if it *)
(* had textmode coordinates for 80 column by 24 rows.                  *)
PROCEDURE GGOTOXY;
VAR XPOS,
    YPOS : INTEGER;
BEGIN
  XPOS := X * 7;
  YPOS := 191 - (Y * 8);
  MOVETO(XPOS,YPOS);
END;  (* ggotoxy *)


(* set up the grafix mode *)
PROCEDURE INITFORGRAFIX;
BEGIN
  INITGRAFIX;
  GRAFIXMODE(BW560,1);
  VIEWPORT(0,559,0,191);
  FILLCOLOR(WHITE);
  PENCOLOR(BLACK);
  FILLPORT;
END;  (* initforgrafix *)


(* delay, do nothing for x seconds *)
PROCEDURE DELAY;
VAR START,
    STOP : REAL;
    ELASPED : INTEGER;
BEGIN
  START := TIME;
  REPEAT
    STOP := TIME;
    ELASPED := TRUNC(ABS(STOP - START))
  UNTIL ELASPED >= SECONDS;
END;      (* delay *)



(* session header *)
PROCEDURE SESSIONHEADER;
VAR SCHAR : CHAR;
BEGIN
  PAGE(OUTPUT);
  INITFORGRAFIX;
  GRAFIXON;
  GLOAD('CATDATA:CATPIC.FOTO');
  GGOTOXY(55,3);
  GWRITESTR(VERSION);
  GGOTOXY(1,23);
  GWRITESTR('Press the <YES> key to begin session ');
  if not ghost then
  begin
```

```
    SCHAR :=
      GETCHAR([CHR(YESKEY),CHR(ESC)],TRUE,FALSE,TRUE);
    IF SCHAR = CHR(ESC) THEN
    BEGIN
      SETCHAIN('CATDATA:CATPROJECT');
      TEXT80MODE;
      EXIT(PROGRAM);
    END;
  end;

  PAGE(OUTPUT);
  TEXTON;

  DELAY(1);

  (* set normal flow *)
  FLOWCODE := CONTINUE;

  (* mark time at beginning of session *)
  SESSSTART := TIME;

END;   (* session header *)



(* session end message *)
PROCEDURE CLOSINGMESSAGE;
VAR I,J,K : INTEGER;
BEGIN

  TEXT40MODE;

  (* mark total time spent at console *)
  EXAMINEE.TOTTIMECONSOLE := TRUNC(TIME - SESSSTART);

  (* save time spent in computer orientation *)
  EXAMINEE.ORIENTATIONTIME := OTIME;

  (* save current examinee data *)
  UPDATEEXAMINEE(ERECNUM);

  PAGE(OUTPUT);
  GOTOXY(4,9);
  WRITE('This completes the testing session.');
  GOTOXY(13,11);
  WRITE('  Thank you.  ');
  GOTOXY(6,13);
  WRITE('Please report to the proctor.');
  DELAY(8);
  (* kill some time *)
  IF (EXAMINEE.LASTTEST <= 10) OR (DEMOFLAG) THEN
    DELAY(8);

END;  (* closing message *)


(**** display a message/wait for a keystroke ****)
PROCEDURE STALL;
VAR STALLCHAR : CHAR;
BEGIN
  WRITE(' Press the <YES> key    ');
  if not ghost then
  begin
    STALLCHAR :=
      GETCHAR([CHR(YESKEY),CHR(ESC)],TRUE,FALSE,TRUE);
    IF STALLCHAR = CHR(ESC) THEN
    BEGIN
      SETCHAIN('CATDATA:CATPROJECT');
      TEXT80MODE;
      EXIT(PROGRAM);
    END;
  end;
END;  (* stall *)
```

```
(* reads the item text file & displays item text *)
PROCEDURE DECODEPRINT;
CONST BLOCKSOUT = 4;
VAR X,
    Y,
    B,
    CURRPTR,
    CURRBLK,
    CHARCODE,
    CHARCNT : INTEGER;

    BADIO : BOOLEAN;


    (* reads a block from disk into the item ascii buffer *)
    PROCEDURE READITEMBLOCK(WHICHBLOCK : INTEGER);
    VAR BLOCKSTRANSFERRED,
        ERRNUM : INTEGER;

        BADIO : BOOLEAN;
    BEGIN
      BADIO := FALSE;
      RESET(ITEMTEXT,TEXTNAME);  (* question text *)
      BLOCKSTRANSFERRED :=
        BLOCKREAD(ITEMTEXT,TRIX.ASCIIBUF,BLOCKSOUT,WHICHBLOCK);
      BADIO := ((BLOCKSTRANSFERRED < 1) OR (IORESULT <> 0));
      ERRNUM := IORESULT;
      CLOSE(ITEMTEXT,LOCK);
      IF BADIO THEN
      BEGIN
        WRITELN;WRITELN;
        WRITE('Block read io error # ',ERRNUM);
        STALL;
        EXIT(PROGRAM);
      END;
    END;

    (* return the next code in ascii file *)
    FUNCTION BUFCODE : INTEGER;
    BEGIN
      BUFCODE := TRIX.ASCIIBUF[CURRPTR];
      CURRPTR := CURRPTR + 1;
      IF CURRPTR > MAXITEMBUF THEN
      (* end of block/get next block and reset byte ptr *)
      BEGIN
        CURRBLK := CURRBLK + BLOCKSOUT;
        READITEMBLOCK(CURRBLK);
        CURRPTR := 0;
      END;
    END;  (* bufcode *)


BEGIN  (* decode print *)
  PAGE(OUTPUT);  (* clear the screen *)
  READITEMBLOCK(BLOCKNUM);
  (* set block/byte ptrs *)
  CURRPTR := BLOCKPTR;
  CURRBLK := BLOCKNUM;
  FILLCHAR(LINEBUF[0],80,' ');
  (* read bytes from the buffer *)
  REPEAT

    (* get char from block buffer *)
    CHARCODE := bufcode;

    CASE CHARCODE OF
      GOTOFLAG : BEGIN  (* move cursor *)
```

```
          (* next two bytes after flag are x,y coord *)
          X := BUFCODE;
          Y := BUFCODE;
          CHARCNT := BUFCODE;
          IF (CURRPTR + CHARCNT - 1) > MAXITEMBUF THEN
          BEGIN
           B := (MAXITEMBUF + 1) - CURRPTR;
           MOVELEFT(TRIX.ASCIIBUF[CURRPTR],LINEBUF[X],B);
           X := X + B;
           B := CHARCNT - B;
           CURRBLK := CURRBLK + BLOCKSOUT;
           READITEMBLOCK(CURRBLK);
           CURRPTR := 0;
           MOVELEFT(TRIX.ASCIIBUF[CURRPTR],LINEBUF[X],B);
           CURRPTR := CURRPTR + B;
          END
          ELSE
          BEGIN
           MOVELEFT(TRIX.ASCIIBUF[CURRPTR],LINEBUF[X],CHARCNT);
           CURRPTR := CURRPTR + CHARCNT;
           IF CURRPTR > MAXITEMBUF THEN
           BEGIN
            CURRBLK :=  CURRBLK + BLOCKSOUT;
            CURRPTR := 0;
            READITEMBLOCK(CURRBLK);
           END;
          END;
          GOTOXY(0,Y);
          WRITE(LINEBUF);
          FILLCHAR(LINEBUF[0],80,' ');
         END;
   PAGEFLAG : BEGIN  (* wait for keystroke to see rest of text *)
          GOTOXY(1,21);
          STALL;
          PAGE(OUTPUT);
         END;
   ENDITEM  : ;

  END;
 UNTIL CHARCODE = ENDITEM; (* until end flag hit *)
END;  (* decodeprint *)



(* blank out lines *)
PROCEDURE BLANKLINES;
VAR I : INTEGER;
BEGIN

 (* begin at *)

 (* blank out so many lines *)
 FOR I := 0 TO (COUNT-1) DO
 BEGIN
  GOTOXY(0,START+I);
  IF FORTYCOLUMN  THEN
  WRITE('                   ') (* ORIG *)
(*AD*)
  ELSE WRITE('                                ');
 END;

 (* leave the cursor at this line *)
 GOTOXY(0,ENDCURSOR);
END;  (* blanklines *)


(* given a question code, this function returns the location *)
(* of the question data, & text pointers.                    *)
FUNCTION SLOT;
VAR INDEX : INTEGER;
  FOUND : BOOLEAN;
BEGIN
```

29

```
  INDEX := MAXSAMPLES + 1;
  FOUND := FALSE;
  REPEAT
    IF DIRECTORY.ITEMCODE[INDEX] = CODE THEN
      FOUND := TRUE
    ELSE
      INDEX := INDEX + 1;
  UNTIL (INDEX > MAXITEMPOOL) OR (FOUND);
  IF FOUND THEN
    SLOT := INDEX
  ELSE
    SLOT := NIL;
END;  (* slot *)


(* returns record # of question data file *)
(* no collisions                          *)
FUNCTION HASH;
BEGIN
HASH :=
  (CURRTEST * MAXITEMPOOL)
  + KEY + CURRTEST;
END;  (* hash *)



(* read in a string representation of an integer an& return integer value *)
FUNCTION GETINTSTR;
VAR PLACE,
    VALUE,
    I         : INTEGER;
    PROCTCALL : BOOLEAN;
BEGIN
  FILLCHAR(LINEBUF[0],4,' ');
  proctcall := false;
  if ghost then
    value := (random mod 1000)
  else
  begin
    REPEAT
      (* read a maximum of four digits *)
      FILLBUF(4,DIGITS + [CHR(HELPKEY)],TRUE);  (* read in 5 digits *)
    UNTIL LINEBUF[0] <> ' ';
    VALUE := 0;
    PLACE := 1;
    PROCTCALL := FALSE;
    I := 3;

    (* convert string to integer *)
    REPEAT
      IF LINEBUF[I] IN DIGITS THEN
      BEGIN
        VALUE := VALUE + ((ORD(LINEBUF[I]) - ASCIIOFFSET) * PLACE);
        PLACE := PLACE * 10;
      END
      ELSE
        IF LINEBUF[I] = CHR(HELPKEY) THEN
          PROCTCALL := TRUE;
      I := I - 1;
    UNTIL (I < 0) OR (PROCTCALL);
    FILLCHAR(LINEBUF[0],4,' ');

  end;

    (* if proctor was called during key in then flag it *)
  IF PROCTCALL THEN
    GETINTSTR := NIL
  ELSE
    GETINTSTR := VALUE;
END;   (* getintstr *)
```

```
(* does a write to graphics screen for char values *)
PROCEDURE GWRITECHR;
VAR C : STRING;
BEGIN
  C := ' ';
  C[1] := GCHR;
  UNITWRITE(3,C[1],1,0,12);
END;  (* gwritechr *)


(* does a write to graphics screen for integer values *)
PROCEDURE GWRITEINT;
VAR X,Z : INTEGER;
    DIGITSTR,STR : STRING;
    NEGATIVE : BOOLEAN;
    C : CHAR;
BEGIN
  NEGATIVE := FALSE;
  Z := GINT;
  IF GINT < 0 THEN
  BEGIN
    NEGATIVE := TRUE;
    Z := -GINT;
  END;
  DIGITSTR := ' ';
  STR := '';
  REPEAT
    X := Z MOD 10;
    C := CHR(X+48);
    DIGITSTR[1] := C;
    STR := CONCAT(DIGITSTR,STR);
    Z := Z DIV 10;
  UNTIL Z <= 0;
  IF NEGATIVE THEN
    STR := CONCAT('-',STR);
  UNITWRITE(3,STR[1],LENGTH(STR),0,12);
END;   (* gwriteint *)


(* do  a writeln to the graphics screen *)
PROCEDURE GWRITELN;
BEGIN
  MOVETO(0,YLOC-8);
END;  (* gwriteln *)


(* does a stall to the graphics screen *)
PROCEDURE GSTALL;
BEGIN
  GWRITESTR('Press the <YES> key ');
  if not ghost then
    IF GETCHAR([CHR(YESKEY)],TRUE,FALSE,TRUE) = CHR(YESKEY) THEN;
END;    (* gstall *)


(* graphics equivalent of the procedure getchar *)
FUNCTION GGETCHAR;
VAR MASK : PACKED ARRAY[0..0] OF CHAR;
    M : STRING;
    ASCII : INTEGER;
BEGIN
  M := ' ';
  IF FLUSHQUEUE THEN UNITCLEAR(2);
  REPEAT
    UNITREAD(2,MASK,1);
    ASCII := ORD(MASK[0]);
    CASE ASCII OF
      AKEY : MASK[0] := 'A';
      BKEY : MASK[0] := 'B';
      CKEY : MASK[0] := 'C';
      DKEY : MASK[0] := 'D';
      EKEY : MASK[0] := 'E';
```

31

```
    END;
    IF BEEP AND NOT (MASK[0] IN OKSET) THEN SQUAWK;
  UNTIL MASK[0] IN OKSET;
  IF ECHO AND (MASK[0] IN [CHR(32)..CHR(126)]) AND
     (NOT (MASK[0] IN CONTROLKEYS)) THEN
  BEGIN
    M[1] := MASK[0];
    UNITWRITE(3,M[1],1,0,12);
  END;
  GGETCHAR := MASK[0];
END;    (* ggetchar *)


(* blanklines on the graphics screen, treat as if in textmode  *)
PROCEDURE GBLANKLINES;
VAR TOP,
    BOTTOM : INTEGER;
BEGIN
  TOP := 191 - (STARTBLANK * 8);
  BOTTOM := 191 - ((STARTBLANK + BLANKTHISMANY) * 8);
  VIEWPORT(0,559,BOTTOM,TOP);
  FILLPORT;
  VIEWPORT(0,559,0,191);
  GGOTOXY(0,LEAVECURSOR);
END;    (* gblanklines *)



(* read in a string and save in temporary buffer, graphics version *)
PROCEDURE GFILLBUF;
VAR I : INTEGER;
    IDCHAR : CHAR;
    B : STRING;
BEGIN
  B := ' ';
  I := 0;
  REPEAT
    IF I > (CHARCNT - 1) THEN
      (* maximum char typed in/allow only backspace or return *)
      IDCHAR :=
        GGETCHAR([CHR(ERASEKEY),CHR(YESKEY)],TRUE,FALSE,TRUE)
    ELSE
    BEGIN
      (* get a character *)
      IDCHAR :=
        GGETCHAR(OKSET + [CHR(ERASEKEY),CHR(YESKEY)],TRUE,FALSE,TRUE);
      IF (IDCHAR IN OKSET) AND (IDCHAR <> CHR(YESKEY)) THEN
      (* save visible character *)
      BEGIN
        GWRITECHR(IDCHAR);
        LINEBUF[I] := IDCHAR;
        I := I + 1;
      END;
    END;
    IF IDCHAR = CHR(ERASEKEY) THEN
    (* backspace key hit *)
    BEGIN
      IF I = 0 THEN
        SQUAWK
      ELSE
      BEGIN
        (* move cursor back one space *)
        MOVETO(XLOC-7,YLOC);
        I := I - 1;
        IF ERASE THEN
        (* blank out backspaced character *)
        BEGIN
          UNITWRITE(3,B[1],1,0,12);
          MOVETO(XLOC-7,YLOC);
          LINEBUF[I] := ' ';
        END;
      END;
```

32

```
    END;
  UNTIL IOCHAR = CHR(YESKEY);
END;    (* gfillbuf *)


(* display question to graphics screen *)
PROCEDURE GDECODEPRINT;
VAR X,Z : INTEGER;
    VNAME,FNAME,DIGITSTR,STR : STRING;
    C : CHAR;


    (* reads the item text file and displays the graphics *)
    PROCEDURE DECODEGRAF;
    VAR X,
        Y,
        X1,Y1,
        CURRPTR,
        CURRBLK,
        DOTCNT : INTEGER;

        (* reads a block from disk into the item a cii buffer *)
        PROCEDURE READITEMBLOCK(WHICHBLOCK : INTEGER);
        VAR BLOCKSTRANSFERRED : INTEGER;
            BADIO : BOOLEAN;
        BEGIN
          BADIO := FALSE;
          RESET(ITEMTEXT,FNAME);
          BLOCKSTRANSFERRED := BLOCKREAD(ITEMTEXT,TRIX.ITEMBUF,4,WHICHBLOCK);
          BADIO := ((BLOCKSTRANSFERRED < 4) OR (IORESULT <> 0));
          CLOSE(ITEMTEXT,LOCK);
          IF BADIO THEN
          BEGIN
            WRITELN;
            WRITELN;
            WRITE(' Block ', WHICHBLOCK,' write error.');
            WRITELN;
            READLN;
            EXIT(PROGRAM);
          END;
        END;  (* readitemblock  *)


        (* reads the item text file & displays item text *)
        PROCEDURE DECODEPRINT;
        VAR X,
            Y,
            BYTECNT,
            CHARCODE : INTEGER;



            (* return the next code in ascii file *)
            FUNCTION BUFCODE : INTEGER;
            BEGIN
              BUFCODE := TRIX.ASCIIBUF[CURRPTR];
              CURRPTR := CURRPTR + 1;
              IF CURRPTR > 2047 THEN
              (* end of block/get next block and reset byte ptr *)
                BEGIN {1}
                  CURRBLK := CURRBLK + 4;
                  READITEMBLOCK(CURRBLK);
                  CURRPTR := 0;
                END;  {1}
            END;  (* bufcode *)


        BEGIN  (* decode print *)
          (* read bytes from the buffer *)
          REPEAT
```

```
       (* get char from block buffer *)
       CHARCODE := bufcode;

       CASE CHARCODE OF
         GOTOFLAG : BEGIN (1)     (* move cursor *)
                      (* next two bytes after flag are x,y coord *)
                      X := BUFCODE;
                      Y := BUFCODE;
                      BYTECNT := BUFCODE;
                      GGOTOXY(X,Y);
                    END;   (1)
         ENDITEM  : ;

       END;

       IF (CHARCODE >= 32) AND (CHARCODE <= 126) THEN
         GWRITECHR(CHR(CHARCODE));

     UNTIL CHARCODE = ENDITEM; (* until end flag hit *)
   END;  (* decodeprint *)


BEGIN  (* decode graf *)

  READITEMBLOCK(0);

  CURRBLK := 0;
  CURRPTR := 0;


  (* decode the xlines *)
  REPEAT
    IF CURRPTR > 1021 THEN
    (* end of block/get new block *)
    BEGIN
      CURRBLK := CURRBLK + 4;
      READITEMBLOCK(CURRBLK);
      CURRPTR := 0;
    END;
    X := TRIX.ITEMBUF[CURRPTR];
    IF X >= 0 THEN
    BEGIN
      Y := TRIX.ITEMBUF[CURRPTR + 1];
      MOVETO(X,Y);
      DOTCNT := TRIX.ITEMBUF[CURRPTR + 2];
      LINEREL(DOTCNT,0);
      CURRPTR := CURRPTR + 3;
    END;
  UNTIL X < 0;

  CURRPTR := CURRPTR + 1;

  (* decode the ylines *)
  REPEAT
    IF CURRPTR > 1021 THEN
    (* end of block/get new block *)
    BEGIN
      CURRBLK := CURRBLK + 4;
      READITEMBLOCK(CURRBLK);
      CURRPTR := 0;
    END;
    X := TRIX.ITEMBUF[CURRPTR];
    IF X >= 0 THEN
    BEGIN
      Y := TRIX.ITEMBUF[CURRPTR + 1];
      MOVETO(X,Y);
      DOTCNT := TRIX.ITEMBUF[CURRPTR + 2];
      LINEREL(0,DOTCNT);
      CURRPTR := CURRPTR + 3;
    END;
```

```
       UNTIL X < 0;

       CURRPTR := CURRPTR + 1;


       (* decode the diagonals *)
       REPEAT
         IF CURRPTR > 1020 THEN
         (* end of block/get new block *)
         BEGIN
           CURRBLK := CURRBLK + 4;
           READITEMBLOCK(CURRBLK);
           CURRPTR := 0;
         END;
         X := TRIX.ITEMBUF[CURRPTR];
         IF X >= 0 THEN
         BEGIN
           Y := TRIX.ITEMBUF[CURRPTR + 1];
           X1 := TRIX.ITEMBUF[CURRPTR + 2];
           Y1 := TRIX.ITEMBUF[CURRPTR + 3];
           MOVETO(X,Y);
           LINETO(X1,Y1);
           CURRPTR := CURRPTR + 4;
         END;
       UNTIL X < 0;

       CURRPTR := CURRPTR + 1;

       (* decode the dots *)
       REPEAT
         IF CURRPTR > 1022 THEN
         (* end of block/get new block *)
         BEGIN
           CURRBLK := CURRBLK + 4;
           READITEMBLOCK(CURRBLK);
           CURRPTR := 0;
         END;
         X := TRIX.ITEMBUF[CURRPTR];
         IF X >= 0 THEN
         BEGIN
           Y := TRIX.ITEMBUF[CURRPTR + 1];
           DOTAT(X,Y);
           CURRPTR := CURRPTR + 2;
         END;
       UNTIL X < 0;

       CURRPTR := CURRPTR + 1;
       CURRPTR := CURRPTR * 2;

       DECODEPRINT;

     END;   (* decodegraf *)


BEGIN
  FILLPORT;
  GRAFIXON;
  DIGITSTR := ' ';
  STR := '';
  Z := ITEMCODE;
  REPEAT
    X := Z MOD 10;
    C := CHR(X+48);
    DIGITSTR[1] := C;
    STR := CONCAT(DIGITSTR,STR);
    Z := Z DIV 10;
  UNTIL Z <= 0;
  DIGITSTR[1] := CHR(SUBTESTNUM+65);
  C := DIGITSTR[1];
  IF SAMPLEQUESTION THEN
  BEGIN
    FNAME := CONCAT('/CATFOTO/',DIGITSTR,'DIR/G',DIGITSTR,'SQ',STR,'.FOTO');
    GLOAD(FNAME);
  END
  ELSE
    IF ITEMINFO.DUMMY1 = COMPRESSED THEN
    BEGIN
      FNAME := CONCAT('/CATFOTO/',DIGITSTR,'DIR/G',DIGITSTR,STR,'.DATA');
      DECODEGRAF;
    END
    ELSE
    BEGIN
      FNAME := CONCAT('/CATFOTO/',DIGITSTR,'DIR/G',DIGITSTR,'D',STR,'.FOTO');
      GLOAD(FNAME);
    END;
END;    (* gdecodeprint *)


(* read in a string representation of an integer and return integer value *)
FUNCTION GGETINTSTR;
VAR PLACE,
    VALUE,
    I       : INTEGER;
    PROCTCALL : BOOLEAN;
BEGIN
  FILLCHAR(LINEBUF[0],4,' ');
  proctcall := false;
  if ghost then
    value := (random mod 1000)
  else
  begin

    REPEAT
      (* read a maximum of four digits *)
      GFILLBUF(4,DIGITS + [CHR(HELPKEY)],TRUE);  (* read in 5 digits *)
    UNTIL LINEBUF[0] <> ' ';
    VALUE := 0;
    PLACE := 1;
    PROCTCALL := FALSE;
    I := 3;

    (* convert string to integer *)
    REPEAT
      IF LINEBUF[I] IN DIGITS THEN
      BEGIN
        VALUE := VALUE + ((ORD(LINEBUF[I]) - ASCIIOFFSET) * PLACE);
        PLACE := PLACE * 10;
      END
      ELSE
        IF LINEBUF[I] = CHR(HELPKEY) THEN
          PROCTCALL := TRUE;
      I := I - 1;
    UNTIL (I < 0) OR (PROCTCALL);
    FILLCHAR(LINEBUF[0],4,' ');
  end;

  (* if proctor was called during key in then flag it *)
  IF PROCTCALL THEN
    GGETINTSTR := NIL
  ELSE
    GGETINTSTR := VALUE;
END;    (* ggetintstr *)



(* initializes the cat system *)
PROCEDURE INITCAT;
TYPE DATE = RECORD
              MONTH,
              DAY,
              YEAR : PACKED ARRAY[0..1] OF CHAR;
            END;
```

```
VAR CATDATE : DATE;
    FILEDATE : FILE OF DATE;

    DATEOK : BOOLEAN;

BEGIN
  PAGE(OUTPUT);
  trace := false;
  DIGITS := ['0'..'9'];
  CONTROLKEYS := [CHR(HELPKEY),CHR(NOKEY),CHR(YESKEY),CHR(ERASEKEY)];
  KEYPAD := DIGITS + CONTROLKEYS;
  RANDOMIZE;
  GRAFIX := FALSE;
  INITFORGRAFIX;
  FORTYCOLUMN := TRUE;
  REVERSEVIDEO := TRUE;
  DEMOFLAG := TRUE;
  SKIP := FALSE;
  SKIPFAM:= FALSE;
  ghost := false;
  ABLOOP := MINGTHETA;
  RIGHT := TRUE;
  TEXTON;
  TEXT40MODE;
  INVERSE;

  (* get system set up/configuration *)
  LOADPARAMS;

  (* get the system date *)
  RESET(FILEDATE,'CATDATA:CATDATE.DATA');
  SEEK(FILEDATE,0);
  GET(FILEDATE);
  CATDATE := FILEDATE^;
  DATEOK := FALSE;
  REPEAT
    PAGE(OUTPUT);
    GOTOXY(1,9);
    WRITELN('The CAT System Date is ',CATDATE.MONTH,' / ',
                                      CATDATE.DAY,' / ',
                                      CATDATE.YEAR);
    WRITELN;
    WRITELN(' Do you want to change the date ?');
    WRITE(' Press <YES> or <NO> : ');
    IF (GETCHAR([CHR(YESKEY),CHR(NOKEY)],TRUE,TRUE,TRUE) = CHR(NOKEY))  THEN
      DATEOK := TRUE
    ELSE
    BEGIN
      WRITELN;
      WRITELN;
      WRITE(' Enter month, then press <YES> : ');
      FILLCHAR(LINEBUF[0],2,' ');
      FILLBUF(2,DIGITS,TRUE);
      MOVELEFT(LINEBUF[0],CATDATE.MONTH[0],2);
      WRITELN;
      WRITE(' Enter the day, then press <YES> : ');
      FILLCHAR(LINEBUF[0],2,' ');
      FILLBUF(2,DIGITS,TRUE);
      MOVELEFT(LINEBUF[0],CATDATE.DAY[0],2);
      WRITELN;
      WRITE(' Enter the year, then press <YES> : ');
      FILLCHAR(LINEBUF[0],2,' ');
      FILLBUF(2,DIGITS,TRUE);
      MOVELEFT(LINEBUF[0],CATDATE.YEAR[0],2);
    END;
  UNTIL DATEOK;
  SEEK(FILEDATE,0);
  FILEDATE^ := CATDATE;
  PUT(FILEDATE);
  CLOSE(FILEDATE,LOCK);
  MOVELEFT(CATDATE.MONTH[0],SYSTEMDATE[0],2);
  MOVELEFT(CATDATE.DAY[0],SYSTEMDATE[2],2);
  MOVELEFT(CATDATE.YEAR[0],SYSTEMDATE[4],2);
```

```
END;    (* initcat *)
```

```
(*************************************************************************)
(*                                                                       *)
(*      Textfile : ADMIN.DIR/A.2UTL.TEXT         Volume : TFILES         *)
(*      Codefile : ADMIN.CODE ('Include' file)   Volume : CATDATA        *)
(*                                                                       *)
(*************************************************************************)
(* File Last Modified :  AUG 16, 1983                        NPROC       *)
(*************************************************************************)

(* Read in a string and save in a temporary buffer can specify maximum char   *)
(* you can type in.  'Help' key is 'live'.  If someone types the help-key in *)
(* any part of string, control passed to the proctor and code is passed back *)
(* to control further flow.  *)

PROCEDURE PFILLBUF;
VAR I,
    FILLCODE : INTEGER;
    IDCHAR : CHAR;
BEGIN
  I := 0;    (* initialize count of characters *)
  REPEAT
    IF I > (CHARCNT-1)
      THEN
        BEGIN {1}
          (* maximum char typed in/allow only backspace or return *)
          IDCHAR :=
            GETCHAR([CHR(HELPKEY),CHR(ERASEKEY),CHR(YESKEY)],TRUE,FALSE,TRUE);

          (* helpkey pressed *)
          IF IDCHAR = CHR(HELPKEY)
            THEN
                BEGIN {2}
                  FILLCODE := 1;
                  CALLPROCT(FILLFROM,FILLCODE);

                  (* pass back negative flowcode if proctor called *)
                  FFLOW := -FILLCODE;
                  EXIT(PFILLBUF);
                END;  {2}
        END   {1}
      ELSE
        BEGIN {3}
          (* get a character *)
          IDCHAR :=
            GETCHAR(OKSET + [CHR(HELPKEY),CHR(YESKEY),
                    CHR(ERASEKEY)], TRUE, TRUE, TRUE);
          IF IDCHAR = CHR(HELPKEY)
            THEN
                BEGIN {4}
                  FILLCODE := 1;
                  CALLPROCT(FILLFROM,FILLCODE);

                  (* passback negative flowcode to signal proctor call *)
                  FFLOW := -FILLCODE;
                  EXIT(PFILLBUF);
                END;  {4}
          IF (IDCHAR IN OKSET) AND (IDCHAR <> CHR(YESKEY))
            THEN
                (* save visible character *)
                BEGIN {5}
                  LINEBUF[I] := IDCHAR;
                  I := I + 1;
                END;  {5}
        END;  {3}
    IF IDCHAR = CHR(ERASEKEY)
      THEN
        (* back space key hit *)
        BEGIN {6}
          IF I = 0
            THEN  SQUAWK (* no character to backspace over *)
            ELSE
                BEGIN {7}
                  WRITE(CHR(LARROW));  (* move cursor back one *)
```

39

```
                    I := I - 1;          (* adjust string buffer location *)
                    IF ERASE
                      THEN
                          (* blank out backspaced character *)
                          BEGIN {8}
                            WRITE(' ');
                            WRITE(CHR(LARROW));
                            LINEBUF[I] := ' ';
                          END;  {8}
                  END;  {7}
          END;  {6}
  UNTIL IDCHAR = CHR(YESKEY);
END;  (* pfillbuf *)

{-----------------------------------------------------------------------------}

(* Display a message/wait for a keystroke.  Helpkey is 'live'.  Call proctor  *)
(* if pressed.  *)

PROCEDURE PSTALL;
VAR STALLCHAR : CHAR;
    STALLCODE : INTEGER;
BEGIN
  WRITE(' Press the <YES> key    ');
  IF NOT GHOST
    THEN
        BEGIN {1}
          STALLCHAR :=
            GETCHAR([CHR(HELPKEY),CHR(YESKEY),CHR(ESC)],TRUE,FALSE,TRUE);
          IF STALLCHAR = CHR(ESC)
            THEN
                BEGIN {2}
                  SETCHAIN('CATDATA:CATPROJECT');
                  TEXT80MODE;
                  EXIT(PROGRAM);
                END;  {2}
          IF STALLCHAR = CHR(HELPKEY)
            THEN
                BEGIN {3}
                  STALLCODE := 1;
                  CALLPROCTOR(STALLFROM,STALLCODE);

                  (* pass back the negative of the flow code to *)
                  (* signal that the help key was pressed.      *)
                  SFLOW := -STALLCODE;
                END;  {3}
        END;  {1}
END;  (* stall *)

{-----------------------------------------------------------------------------}

(* Get the character response of a multiple choice question *)
FUNCTION GETCHRANSWER;
VAR CHARSELECT : CHAR;
    OK : BOOLEAN;
    GCODE : INTEGER;
    PCRECNUM: BOOLEAN;

BEGIN
  OK := FALSE;
  PCRECNUM := (CURRTEST=4) OR (CURRTEST=6);
  IF PCRECNUM
  THEN GOTOXY(0,23)
  ELSE
  GOTOXY(0,21);
  REPEAT
    REPEAT
      WRITE(' Please enter your answer : ');
      IF GHOST THEN
        BEGIN
          IF GHOSTFLOW = OLD
          THEN
          CHARSELECT := GHOSTKEY
```

40

```
          ELSE
          BEGIN
            IF GHOSTFLOW = AUTO THEN
            GTHETA := ABLOOP;
          CHARSELECT := NEWGHOSTKEY(GTHETA);
          END;
        END
      ELSE
        IF CURRSTRAT = TIMED THEN  (* dont flush the buffer or beep *)
           CHARSELECT := GGETCHAR([ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER,
                                    CHR(HELPKEY)], FALSE,FALSE, FALSE)
        ELSE
           CHARSELECT := GGETCHAR([ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER,
                                    CHR(HELPKEY)], TRUE, FALSE, TRUE);
      IF CHARSELECT = CHR(HELPKEY)
        THEN
            BEGIN {1}
              GCODE := 1;
              CALLPROCTOR(CALLEDFROM,GCODE);

              (* pass back negative flow code if proctor was called *)
              PCODE := -GCODE;
              EXIT(GETCHRANSWER);
            END;  {1}
      IF PCRECNUM THEN
      BLANKLINES(23,1,23)
      ELSE
      BLANKLINES(21,1,21);
    UNTIL CHARSELECT <> CHR(HELPKEY);
    IF PCRECNUM
    THEN
       BEGIN
         WRITE(' Your response was ',CHARSELECT);
         WRITE('          Do you want this answer? ');
       END
    ELSE
       BEGIN
         WRITELN(' Your response was ',CHARSELECT);
         WRITELN;
         WRITE(' Do you want this answer? ');
       END;
    IF GHOST
      THEN OK := TRUE
      ELSE
          BEGIN {2}
            IF GETCHAR([CHR(YESKEY),CHR(NOKEY)],TRUE,FALSE,TRUE) = CHR(NOKEY)
              THEN
                BEGIN
                  IF PCRECNUM
                     THEN BLANKLINES(23,1,23)
                  ELSE
                     BLANKLINES(21,3,21);
                END
              ELSE OK := TRUE;
          END;  {2}
  UNTIL OK;
  GETCHRANSWER := CHARSELECT;
END;   (* getchranswer *)

{-----------------------------------------------------------------------------}

(* get the integer response to a question *)
FUNCTION GETINTANSWER;
VAR GCODE,
    INTSELECT : INTEGER;
    OK : BOOLEAN;
BEGIN {Getintanswer}
  OK := FALSE;
  GOTOXY(0,21);
  REPEAT
    WRITE(' Please enter your answer : ');
    INTSELECT := GETINTSTR;
    IF INTSELECT < 0
```

41

```
      THEN
          BEGIN {1}
            GCODE := 1;
            CALLPROCTOR(CALLEDFROM,GCODE);

            (* pass back negative flowcode to signal proctor call *)
            PCODE := -GCODE;
            EXIT(GETINTANSWER);
          END;  {1}
    BLANKLINES(21,1,21);
  UNTIL INTSELECT >= 0;
  GETINTANSWER := INTSELECT;
END;   (* getintanswer *)

{------------------------------------------------------------------------------}

(* Get seven responses at once, this is for the subtest "coding speed" *)
(* and others. *)
PROCEDURE GETSEVENANSWERS;
VAR ONECHAR : CHAR;
    OK : BOOLEAN;
    ETIME,
    GCODE,
    I  : INTEGER;
    STARTCLOCK,
    STOPCLOCK : REAL;
BEGIN
  I := 0;
  ETIME := 0;
  REPEAT
    I := I + 1;
    OK := FALSE;
    BLANKLINES(21,3,21);
    STARTCLOCK := TIME;
    REPEAT
      REPEAT
        WRITE(' Enter your answer for question ',QSTNUM+I-1,' : ');
        IF GHOST THEN
          ONECHAR := GHOSTKEY
        ELSE
          IF CURRSTRAT = TIMED THEN  (* dont flush buffer *)
             ONECHAR := GETCHAR([ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER,
                                      CHR(HELPKEY)], FALSE, FALSE, FALSE)
          ELSE
             ONECHAR := GETCHAR([ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER,
                                   CHR(HELPKEY)], TRUE, FALSE, TRUE);
        IF ONECHAR = CHR(HELPKEY)
          THEN
              BEGIN {1}
                GCODE := 1;
                CALLPROCTOR(CALLEDFROM,GCODE);
                PCODE := -GCODE;

                (* Exit if leaving subtest and not returning *)
                (* or if just answering a sample.     *)
                IF ((GCODE > 3) AND (GCODE <> 6)) OR
                   ((GCODE = 6) AND (SAMPLEQUESTION))
                     THEN EXIT(GETSEVENANSWERS);

                (* Else, display the question again and examinee *)
                (* must answer it before leaving.               *)
                DECODEPRINT(GBLOCK,GPTR);
              END;  {1}
        BLANKLINES(21,1,21);
      UNTIL ONECHAR <> CHR(HELPKEY);
      WRITELN(' Your response for question ',QSTNUM+I-1,' was ',ONECHAR);
      WRITELN;
      WRITE(' Do you want this answer? ');
      IF GHOST
        THEN OK := TRUE
        ELSE
            BEGIN {2}
              IF GETCHAR([CHR(YESKEY),CHR(NOKEY)], FALSE, FALSE, FALSE) =
                                 CHR(NOKEY)
        THEN BLANKLINES(21,3,21)
        ELSE OK := TRUE;
      END;  (2)
  UNTIL OK;
  A7[I] := ONECHAR;   (* a7 is a variable parameter *)
  IF NOT SAMPLEQUESTION THEN
  BEGIN (3)
   STOPCLOCK := TIME;
   ETIME := ETIME + (TRUNC(ABS(STOPCLOCK - STARTCLOCK)));
   IF (ELASPEDTIME + ETIME) >= MAXTIME
    THEN TIMEOUT := TRUE;
  END;  (3)
 UNTIL (I = QSTCNT) OR (TIMEOUT);

 IF NOT SAMPLEQUESTION THEN PAGE(OUTPUT);


 (* pass back # of questions answered *)
 IF TIMEOUT
  THEN QACNT := 0    (* throw away all questions if timeout occured *)
  ELSE QACNT := I;
 PCODE := 1;  (* signal that answer is valid *)
END;   (* getsevenanswers *)

{-----------------------------------------------------------------------}

(* Read a string and save in a temporary buffer.  Can specify maximum char    *)
(* you can type in.  Help key is 'live'.  If someone types helpkey in any     *)
(* part of string, cotrol passed to proctor and  code is passed back to       *)
(* control further flow.     *)
PROCEDURE GPFILLBUF;
VAR I,
  FILLCODE : INTEGER;
  IOCHAR : CHAR;
  B : STRING;
BEGIN
 B := ' ';
 I := 0;   (* initialize count of characters *)
 REPEAT
  IF I > (CHARCNT-1)
   THEN
     BEGIN (1)
      (* maximum char typed in/allow only backspace or return *)
      IOCHAR :=
       GETCHAR([CHR(HELPKEY),CHR(ERASEKEY),CHR(YESKEY)],TRUE,FALSE,TRUE);

      (* helpkey pressed *)
      IF IOCHAR = CHR(HELPKEY)
       THEN
         BEGIN (2)
          FILLCODE := 1;
          CALLPROCT(FILLFROM,FILLCODE);

          (* pass back negative flowcode if proctor called *)
          FFLOW := -FILLCODE;
          EXIT(GPFILLBUF);
         END;  (2)
     END   (1)
   ELSE
     BEGIN (3)
      (* get a character *)
      IOCHAR :=
       GETCHAR(OKSET + [CHR(HELPKEY),CHR(YESKEY),
           CHR(ERASEKEY)], TRUE, FALSE, TRUE);
      IF IOCHAR = CHR(HELPKEY)
       THEN
         BEGIN (4)
          FILLCODE := 1;
          CALLPROCT(FILLFROM,FILLCODE);

          (* passback negative flowcode to signal proctor call *)
          FFLOW := -FILLCODE;
```

```
                    EXIT(GPFILLBUF);
                  END;  {4}
            IF (IDCHAR IN OKSET) AND (IDCHAR <> CHR(YESKEY))
              THEN
                  (* save visible character *)
                  BEGIN {5}
                    GWRITECHR(IDCHAR);
                    LINEBUF[I] := IDCHAR;
                    I := I + 1;
                  END;  {5}
          END;  {3}
    IF IDCHAR = CHR(ERASEKEY)
      THEN
          (* back space key hit *)
          BEGIN {6}
            IF I = 0
              THEN SQUAWK     (* no character to backspace over *)
              ELSE
                  BEGIN {7}
                    MOVETO(XLOC-7,YLOC);  (* move cursor back one *)
                    I := I - 1;           (* adjust string buffer location *)
                    IF ERASE
                      THEN
                          (* blank out backspaced character *)
                          BEGIN {8}
                            UNITWRITE(3,B[1],1,0,12);
                            MOVETO(XLOC-7,YLOC);
                            LINEBUF[I] := ' ';
                          END;  {8}
                  END;  {7}
          END;  {6}
  UNTIL IDCHAR = CHR(YESKEY);
END;  (* gpfillbuf *)

{-------------------------------------------------------------------------------}

(* get the character responce of a multiple choice question *)
FUNCTION GGETCHRANSWER;
VAR CHARSELECT : CHAR;
    OK : BOOLEAN;
    GCODE : INTEGER;
BEGIN
  OK := FALSE;
  GGOTOXY(0,21);
  REPEAT
    REPEAT
      GWRITESTR(' Please enter your answer : ');
      IF GHOST
        THEN
          BEGIN
            IF GHOSTFLOW = OLD
            THEN  CHARSELECT := GHOSTKEY
            ELSE
              BEGIN
                IF GHOSTFLOW = AUTO
                THEN GTHETA:=ABLOOP;
              CHARSELECT:= NEWGHOSTKEY(GTHETA);
              END;
          END
        ELSE CHARSELECT := GETCHAR([ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER,
                                    CHR(HELPKEY)], TRUE, FALSE, TRUE);
      IF CHARSELECT = CHR(HELPKEY)
        THEN
            BEGIN {1}
              GCODE := 1;
              CALLPROCTOR(CALLEDFROM,GCODE);

              (* pass back negative flow code if proctor was called *)
              PCODE := -GCODE;
              EXIT(GGETCHRANSWER);
            END;  {1}
      GBLANKLINES(21,1,21);
    UNTIL CHARSELECT <> CHR(HELPKEY);
    GWRITESTR(' Your response was ');
    GWRITECHR(CHARSELECT);
    GWRITELN;
    GWRITELN;
    GWRITESTR(' Do you want this answer? ');
    IF GHOST
      THEN OK := TRUE
      ELSE
          BEGIN (2)
            IF GGETCHAR([CHR(YESKEY),CHR(NOKEY)], TRUE, FALSE, TRUE) =
                                                            CHR(NOKEY)
              THEN GBLANKLINES(21,3,21)
              ELSE OK := TRUE;
          END;   (2)
  UNTIL OK;
  GGETCHRANSWER := CHARSELECT;
END;    (* ggetchranswer *)

{----------------------------------------------------------------------------}

(* get the integer response to a question *)
FUNCTION GGETINTANSWER;
VAR GCODE,
    INTSELECT : INTEGER;
    OK : BOOLEAN;
BEGIN
  OK := FALSE;
  GGOTOXY(0,21);
  REPEAT
    GWRITESTR(' Please enter your answer : ');
    INTSELECT := GGETINTSTR;
    IF INTSELECT < 0
      THEN
          BEGIN (1)
            GCODE := 1;
            CALLPROCTOR(CALLEDFROM,GCODE);

            (* pass back negative flowcode to signal proctor call *)
            PCODE := -GCODE;
            EXIT(GGETINTANSWER);
          END;   (1)
    GBLANKLINES(21,1,21);
  UNTIL INTSELECT >= 0;
  GGETINTANSWER := INTSELECT;
END;    (* ggetintanswer *)

{----------------------------------------------------------------------------}

(* get seven responses at once, this is for the subtest "coding speed" *)
(* and others. *)
PROCEDURE GGETSEVENANSWERS;
VAR ONECHAR : CHAR;
    OK : BOOLEAN;
    ETIME,
    GCODE,
    I  : INTEGER;
    STARTCLOCK,
    STOPCLOCK : REAL;
BEGIN
  I := 0;
  ETIME := 0;
  REPEAT
    I := I + 1;
    OK := FALSE;
    GBLANKLINES(21,3,21);
    STARTCLOCK := TIME;
    REPEAT
      REPEAT
        GWRITESTR(' Enter your answer for question ');
        GWRITEINT(QSTNUM+I-1);
        GWRITESTR(' : ');
        IF GHOST
          THEN ONECHAR := GHOSTKEY
          ELSE ONECHAR := GGETCHAR([ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER,
                                  CHR(HELPKEY)], TRUE, FALSE, TRUE);
        IF ONECHAR = CHR(HELPKEY)
          THEN
              BEGIN (1)
                GCODE := 1;
                CALLPROCTOR(CALLEDFROM,GCODE);
                PCODE := -GCODE;

                (* exit if leaving subtest and not returning *)
                (* or if we are just answering a sample.     *)
                IF (GCODE > 3) AND (GCODE <> 6)
                  THEN EXIT(GGETSEVENANSWERS);

                (* else display the question again and examinee *)
                (* must answer it before leaving.               *)
                GDECODEPRINT(GTEST,GITEM);
              END;  (1)
        GBLANKLINES(21,1,21);
      UNTIL ONECHAR <> CHR(HELPKEY);
      GWRITESTR(' Your response for question ');
      GWRITEINT(QSTNUM+I-1);
      GWRITESTR(' was ');
      GWRITECHR(ONECHAR);
      GWRITELN;
      GWRITELN;
      GWRITESTR(' Do you want this answer? ');
      IF GHOST
        THEN OK := TRUE
        ELSE
            BEGIN (2)
              IF GGETCHAR([CHR(YESKEY),CHR(NOKEY)], TRUE, FALSE, TRUE) =
                                                                 CHR(NOKEY)
                THEN GBLANKLINES(21,3,21)
                ELSE OK := TRUE;
            END;  (2)
    UNTIL OK;
    A7[I] := ONECHAR;   (* a7 is a variable parameter *)
    IF NOT SAMPLEQUESTION
      THEN
          BEGIN (3)
            STOPCLOCK := TIME;
            ETIME := ETIME + (TRUNC(ABS(STOPCLOCK - STARTCLOCK)));
            IF (ELASPEDTIME + ETIME) >= MAXTIME
              THEN TIMEOUT := TRUE;
          END;  (3)
  UNTIL (I = QSTCNT) OR (TIMEOUT);

  (* pass back # of questions answered *)
  IF TIMEOUT
    THEN QACNT := 0        (* throw away all questions if timeout occured *)
    ELSE QACNT := I;
  PCODE := 1;  (* signal that answer is valid *)
END;   (* ggetsevenanswers *)
```

```
(***************************************************************************)
(*                                                                         *)
(*      Textfile : ADMIN.DIR/A.ESUMM.TEXT         Volume : TFILES          *)
(*      Codefile : ADMIN.CODE ('Include' file)    Volume : CATDATA         *)
(*                                                                         *)
(***************************************************************************)
(*  File last modified : August 5, 1983           NPROC                    *)
(***************************************************************************)

(* this procedure writes out the examinee records in a coded textfile *)
(* in which redundant text is removed.  It is used by another program *)
(* which expects text to be in a certain format, this way it can do   *)
(* a mass analysis of the data.                                       *)
SEGMENT PROCEDURE ESUMMARY;

CONST (* examinee personal data file *)
      PINFONAME = 'CATDATA:EPDATA.DATA';

TYPE  (* examinee personal data record *)
      PINFOREC = RECORD
                   LASTNAME : PACKED ARRAY[0..14] OF CHAR;
                   FIRSTNAME : PACKED ARRAY[0..11] OF CHAR;
                   MINITIAL : CHAR;
                   CURRADDRESS,
                   HOMEOFREC  : PACKED ARRAY[0..1] OF CHAR;
                   CITIZENSHIP : PACKED ARRAY[0..3] OF CHAR;
                   SEX : CHAR;
                   POPGROUP : PACKED ARRAY[0..4] OF CHAR;
                   ETHNIC : PACKED ARRAY[0..1] OF CHAR;
                   MARITAL : CHAR;
                   DEPENDANTS : PACKED ARRAY[0..1] OF CHAR;
                   BIRTHDATE : PACKED ARRAY[0..7] OF CHAR;
                   EDUCATION : PACKED ARRAY[0..2] OF CHAR;
                   TESTID : PACKED ARRAY[0..2] OF CHAR;
                   AFQT : PACKED ARRAY[0..1] OF CHAR;
                   ASVAB  : PACKED ARRAY[0..43] OF CHAR;
                   ENLISTDATE,
                   ACTSERDATE  : PACKED ARRAY[0..7] OF CHAR;
                   ENL : PACKED ARRAY[0..4] OF CHAR;
                   AFEES : PACKED ARRAY[0..1] OF CHAR;
                   BOS : PACKED ARRAY[0..1] OF CHAR;
                   POSTASVAB : PACKED ARRAY[0..43] OF CHAR;
                   STESTORDER : PACKED ARRAY[0..59] OF CHAR;
                 END;


VAR TNUM,
    MAXLINES,
    DUM,
    LINESOUT,
    TESTCOUNT,
    K,
    RSLOT,
    MINUTES,
    SECONDS : INTEGER;


    TNAME,
    ENAME,
    FNAME : STRING;

    SUMTIME,
    SAVETIME : REAL;

    (* examinee personal data *)
    PINFO : PINFOREC;
    PINFOFILE : FILE OF PINFOREC;
```

```
(* display examinee personal data *)
PROCEDURE SHOWEXAMINEE;
VAR J : INTEGER;


    (* display some data *)
    PROCEDURE D1;
    BEGIN
      WITH PINFO DO
      BEGIN
        WRITELN(DEST);
        WRITE(DEST,LASTNAME,' ');
        WRITE(DEST,FIRSTNAME,' ');
        WRITE(DEST,MINITIAL);
        WRITE(DEST,EXAMINEE.ID);
        WRITE(DEST,CURRADDRESS);
        WRITE(DEST,HOMEOFREC);
        WRITE(DEST,CITIZENSHIP);
        WRITE(DEST,SEX);
        WRITELN(DEST,POPGROUP);
             (* end of first line of compacted data *)
        WRITE(DEST,ETHNIC);
        WRITE(DEST,MARITAL);
        WRITE(DEST,DEPENDANTS);
        WRITE(DEST,BIRTHDATE);
      END;
    END;  (* d1 *)

BEGIN  (* display examinee *)
  D1;
  WITH PINFO DO
  BEGIN
    WRITE(DEST,EDUCATION);
    WRITE(DEST,TESTID);
    WRITE(DEST,AFQT);
    WRITELN(DEST,ASVAB);
         (* end of second line of compact data *)
    WRITE(DEST,ENLISTDATE);
    WRITE(DEST,ACTSERDATE);
    WRITE(DEST,ENL);
    WRITELN(DEST,AFEES);
         (* end of third line of compact data *)
    WRITE(DEST,POSTASVAB);
    WRITE(DEST,BOS);
    WRITE(DEST,STESTORDER[0],STESTORDER[1]);
    WRITELN(DEST,EXAMINEE.DATE);
         (* end of fourth line of compact data *)
  END;
END;  (* showexaminee *)



(* check whether coding speed or numerical ops subtest
   needed to see if time info will appear at correct line
   for SPSS *)
PROCEDURE CHECKTIMEOUT;
BEGIN
  (* Test number minus one.  ie:
     5. Numerical Operations         12. Numerical Operations III
     6. Coding Speed                 13. Numerical Operations IV
    10. Coding Speed II              14. Coding Speed III
    11. Numerical Operations II      15. Coding Speed IV *)
  IF EXAMINEE.TESTORDER[TINDEX] IN [5,6,10,11,12,13,14,15] THEN
  BEGIN
    IF EXAMINEE.TESTORDER[TINDEX] IN [6,10,14,15] THEN
      MAXLINES := 7 * EXAMINEE.TESTLENGTH[TINDEX]
    ELSE
      MAXLINES := 3 * EXAMINEE.TESTLENGTH[TINDEX]-1;
    IF LINESOUT < MAXLINES THEN
    BEGIN
      FOR DUM := LINESOUT+1 TO MAXLINES DO
        IF EXAMINEE.TESTORDER[TINDEX] IN [6,10,14,15]
        THEN
```

```
          IF DUM MOD 7 = 0 THEN
          BEGIN
            WRITE(DEST,'Time000 ');
            WRITELN(DEST,SAVETIME : 6 : 1);
            SAVETIME := 0.0;
          END
          ELSE
            WRITE(DEST,'Time000 ')
        ELSE
          BEGIN
            IF DUM MOD 3 = 0 THEN
            BEGIN
              WRITE(DEST,'Time000 ');
              WRITELN(DEST,SAVETIME : 6 : 1);
              SAVETIME := 0.0;
            END
            ELSE
              WRITE(DEST,'Time000 ');
            IF DUM=MAXLINES THEN
            BEGIN
              WRITELN(DEST,'        ',SAVETIME : 6 : 1);
              SAVETIME := 0.0;
            END;
          END;
    END
    ELSE
      IF NOT (EXAMINEE.TESTORDER[TINDEX] IN [6,10,14,15]) THEN
        WRITELN(DEST,'        ',SAVETIME : 6 : 1);

  END;
END;   (* checktimeout *)


PROCEDURE TIME;
BEGIN
  MINUTES := TESTS.STTIME DIV 60;
  SECONDS := TESTS.STTIME MOD 60;
  IF MINUTES > 0 THEN
    WRITE(DEST,MINUTES:3,':')
      ELSE
    WRITE(DEST,'   :');
  IF SECONDS < 10 THEN WRITE(DEST,'0',SECONDS)
    ELSE
  WRITE(DEST,SECONDS:2);
  MINUTES := TESTS.STINSTRTIME DIV 60;
  SECONDS := TESTS.STINSTRTIME MOD 60;
  IF MINUTES > 0 THEN
    WRITE(DEST,MINUTES:3,':')
      ELSE
    WRITE(DEST,'   :');
  IF SECONDS < 10 THEN WRITE(DEST,'0',SECONDS)
    ELSE
  WRITE(DEST,SECONDS:2);
  WRITE(DEST,TESTS.STPROCTCALLS:2);
      (*  line 6 of compact data *)
END; (*TIME*)


(* send detailed or simple feedback to printer or screen *)
PROCEDURE OUTPUTRESULTS;
VAR J,I,K : INTEGER;
    SEVEN : PACKED ARRAY[1..7] OF CHAR;


 (* get info and write header *)
 PROCEDURE INFOHEADER;

   VAR SHORT_NAME : STRING;

 BEGIN
   RESET(FILEDIRECTORY,INDEXNAME);
   SEEK(FILEDIRECTORY,EXAMINEE.TESTORDER[TINDEX]);
   GET(FILEDIRECTORY);
   TNAME := FILEDIRECTORY^.TESTNAME;
   CLOSE(FILEDIRECTORY,LOCK);
   CURRSTRAT  := EXAMINEE.STRATEGY[TINDEX];

   SHORT_NAME := 'WHAT';   (* SET A DEFAULT *)
   CASE EXAMINEE.TESTORDER[TINDEX] OF

      0  :  SHORT_NAME := '  WK';
      1  :  SHORT_NAME := '  GS';
      2  :  SHORT_NAME := '  AR';
      3  :  SHORT_NAME := '  MK';
      4  :  SHORT_NAME := '  PC';
      5  :  SHORT_NAME := 'NO-1';
      6  :  SHORT_NAME := 'CS-1';
      7  :  SHORT_NAME := '  AI';
      8  :  SHORT_NAME := '  MC';
      9  :  SHORT_NAME := '  EI';
     10  :  SHORT_NAME := 'CS-2';
     11  :  SHORT_NAME := 'NO-2';
     12  :  SHORT_NAME := 'NO-3';
     13  :  SHORT_NAME := 'NO-4';
     14  :  SHORT_NAME := 'CS-3';
     15  :  SHORT_NAME := 'CS-4';
     16  :  SHORT_NAME := 'MK-2';
     17  :  SHORT_NAME := '  SI';
     18  :  SHORT_NAME := 'MC-2';
     19  :  SHORT_NAME := 'AR-2';
     20  :  SHORT_NAME := 'EI-2';
   END;  (* CASE *)

   WRITELN(DEST,SHORT_NAME);     (* end of line four of compact data *)

END;  (* infoheader *)


PROCEDURE CTOUT;
BEGIN
  CASE CURRSTRAT OF
    NONE : WRITELN(DEST,'*****************');
    B102222,
    B54321,
    B108642      : BEGIN
                     WRITE(DEST,TESTS.ITEMINFO[I].THETA : 2 : 3);
                     WRITELN(DEST,TESTS.ITEMINFO[I].ERROR : 2 : 3);
                   END;
    TIMED        : BEGIN
                     IF TESTS.ITEMINFO[I].RTYPE <> SEVENCHR THEN
                     BEGIN
                       IF ((I+1) > QUESTIONS) OR
                          ((I+1) > TESTS.NUMITEMS) THEN
                           CHECKTIMEOUT
                       ELSE
                       BEGIN
                        IF TESTS.ITEMINFO[I+1].ITEMNUM <= 0 THEN
                          CHECKTIMEOUT
                         ELSE
                           WRITELN(DEST,
                                 TESTS.ITEMINFO[I].LATENCY : 6 : 1);
                       END;
                     END;
                   END;
  END;
END;    (* ctout *)


BEGIN  (* outputresults *)

  INFOHEADER;
  I := 0;
  SUMTIME := 0.0;
  LINESOUT:=0;
```

```
WHILE (I <= QUESTIONS) AND (I <= TESTS.NUMITEMS) DO
BEGIN
  IF TESTS.ITEMINFO[I].ITEMNUM > 0 THEN
  BEGIN

    IF TESTS.ITEMINFO[I].RTYPE <> SEVENCHR THEN
    BEGIN
      IF TESTS.ITEMINFO[I].ITEMNUM < 10 THEN WRITE(DEST,'000')
       ELSE
      IF TESTS.ITEMINFO[I].ITEMNUM < 100 THEN WRITE(DEST,'00')
       ELSE
      IF TESTS.ITEMINFO[I].ITEMNUM < 1000 THEN WRITE(DEST,'0');

      WRITE(DEST,TESTS.ITEMINFO[I].ITEMNUM,'*');
    END;

    SAVETIME := TESTS.ITEMINFO[I].LATENCY;
    IF CURRSTRAT = TIMED THEN
      SUMTIME := SUMTIME + SAVETIME;

    CASE TESTS.ITEMINFO[I].RTYPE OF
      CHARVALUE : BEGIN
                   CASE TESTS.ITEMINFO[I].RESPONSE OF
                     'A'  :  WRITE(DEST,'1');
                     'B'  :  WRITE(DEST,'2');
                     'C'  :  WRITE(DEST,'3');
                     'D'  :  WRITE(DEST,'4');
                     'E'  :  WRITE(DEST,'5');
                   END(*CASE*);
                   IF TESTS.ITEMINFO[I].CORRECT THEN
                      WRITE(DEST,'1')
                   ELSE
                      WRITE(DEST,'0');
                 END;
      INTVALUE  : BEGIN
                   WRITE(DEST,TESTS.ITEMINFO[I].INTRESPONSE : 1);
                   IF TESTS.ITEMINFO[I].CORRECT THEN
                      WRITE(DEST,'1')
                   ELSE
                      WRITE(DEST,'0');
                 END;
      SEVENCHR  : BEGIN
                   J := TESTS.ITEMINFO[I].ACOUNT;
                   FOR K := 1 TO J DO
                   BEGIN
                     IF TESTS.ITEMINFO[I].ITEMNUM < 10 THEN
                                                WRITE(DEST,'000')
                       ELSE
                     IF TESTS.ITEMINFO[I].ITEMNUM < 100 THEN
                                                WRITE(DEST,'00')
                       ELSE
                     IF TESTS.ITEMINFO[I].ITEMNUM < 1000 THEN
                                                WRITE(DEST,'0');
                     WRITE(DEST,TESTS.ITEMINFO[I].ITEMNUM,K);
                     CASE TESTS.ITEMINFO[I].CHRRESPONSE[K] OF
                       'A'  :  WRITE(DEST,'1');
                       'B'  :  WRITE(DEST,'2');
                       'C'  :  WRITE(DEST,'3');
                       'D'  :  WRITE(DEST,'4');
                       'E'  :  WRITE(DEST,'5');
                     END(*CASE*);
                     IF TESTS.ITEMINFO[I].ACORRECT[K-1] THEN
                       WRITE(DEST,'1')
                     ELSE
                       WRITE(DEST,'0');
                     WRITE(DEST,'*');
                     LINESOUT:=LINESOUT+1;
                   END;  (* for *)

                   TNUM := EXAMINEE.TESTORDER[TINDEX];

                   IF ((I+1) > QUESTIONS) OR
                      ((I+1) > TESTS.NUMITEMS) THEN
                      BEGIN
                        IF ((J = 7) AND (TNUM = 10)) OR
                           ((J = 3) AND (TNUM = 11))  THEN
                        BEGIN
                          WRITELN(DEST,
                                  TESTS.ITEMINFO[I].LATENCY:6:1);

                          IF J = 7 THEN
                            SAVETIME := 0.0;  (* 420.0 - SUMTIME; *)

                          IF J = 3 THEN
                            SAVETIME := 0.0;  (* 180.0 - SUMTIME; *)
                        END;
                        CHECKTIMEOUT;
                      END
                      ELSE
                      BEGIN
                        IF TESTS.ITEMINFO[I+1].ITEMNUM <= 0 THEN
                        BEGIN
                          IF  ((J = 7) AND (TNUM = 10)) OR
                               ((J = 3) AND (TNUM = 11))  THEN
                          BEGIN
                            WRITELN(DEST,
                                    TESTS.ITEMINFO[I].LATENCY:6:1);

                            IF J = 7 THEN
                              SAVETIME := 0.0;  (* 420.0 - SUMTIME; *)
                            IF J = 3 THEN
                              SAVETIME := 0.0;  (* 180.0 - SUMTIME; *)
                          END;

                          CHECKTIMEOUT;
                        END
                        ELSE
                          WRITELN(DEST,
                                  TESTS.ITEMINFO[I].LATENCY:6:1);
                      END;
                    END;
      END;


      CTOUT;

    END;
    I := I + 1;     (* each I outputs a line 5 of compact data *)
  END;      (* WHILE *)
  TIME;

  IF EXAMINEE.PREDASVAB[TINDEX] < 10.0 THEN
  BEGIN
    WRITE(DEST,' '); (* This statement right justifies the PASVAB output *)
    WRITELN(DEST,EXAMINEE.PREDASVAB[TINDEX]:5:2);
  END
  ELSE
  WRITELN(DEST,EXAMINEE.PREDASVAB[TINDEX]:5:2);
        (* end of line 6 of compact data *)

END;    (* outputresults *)

(* loads examinee personal data *)
PROCEDURE LOADPDATA(RECORDNUM : INTEGER);
BEGIN
  RESET(PINFOFILE,PINFONAME);
  SEEK(PINFOFILE,RECORDNUM);
  GET(PINFOFILE);
  PINFO := PINFOFILE^;
  CLOSE(PINFOFILE,LOCK);
END;    (* load personal data *)
```

```
BEGIN   (* examinee summary *)

  LOADPDATA(ERECNUM);

  ENAME := '         ';
  FOR K := 0 TO 8 DO
    ENAME[K+1] := EXAMINEE.ID[K];
  FNAME := CONCAT('E',ENAME);
  FNAME := CONCAT(FNAME,'.TEXT[*]');
  FNAME := CONCAT('QTEXT : ',FNAME);


  REWRITE(DEST,FNAME);

  (* write personal data to file *)
  SHOWEXAMINEE;
  RSLOT := ERECNUM * GMAXSUBTEST;
  FOR TINDEX := 1 TO GMAXSUBTEST DO
  BEGIN
    CURRSTRAT := EXAMINEE.STRATEGY[TINDEX];
    LOADRESULTS(RSLOT);
    IF TESTS.NUMITEMS > 0 THEN
    BEGIN
      WRITELN(DEST);
      OUTPUTRESULTS;
    END;
    RSLOT := RSLOT + 1;
  END;  (* for *)
  MINUTES := EXAMINEE.TOTTIMECONSOLE DIV 60;
  SECONDS := EXAMINEE.TOTTIMECONSOLE MOD 60;
  WRITE(DEST,MINUTES:3,':');
  IF SECONDS < 10 THEN WRITE(DEST,'0',SECONDS)
    ELSE
  WRITE(DEST,SECONDS:2);
  MINUTES := EXAMINEE.ORIENTATIONTIME DIV 60;
  SECONDS := EXAMINEE.ORIENTATIONTIME MOD 60;
  WRITE(DEST,MINUTES:3,':');
  IF SECONDS < 10 THEN WRITE(DEST,'0',SECONDS)
    ELSE
  WRITE(DEST,SECONDS:2);
  WRITELN(DEST,EXAMINEE.NUMPROC:2);
  WRITELN(DEST);
  CLOSE(DEST,LOCK);

END;  (* endsummary *)
```

```
(*************************************************************************)
(*                                                                       *)
(*      Textfile : ADMIN.DIR/A.IO.TEXT          Volume : TFILES          *)
(*      Codefile : ADMIN.CODE ('Include' file)  Volume : CATDATA         *)
(*                                                                       *)
(*************************************************************************)
(*              DEC.  1, 1982                   NPROC                    *)
(*************************************************************************)


(* load the examinee directory *)
PROCEDURE LOADINDEX;
BEGIN
  RESET(EDIR,EINDEX);
  SEEK(EDIR,0);        (* examinee directory always record 0 *)
  GET(EDIR);
  DIR := EDIR^;
  CLOSE(EDIR,LOCK);
END;  (* load index *)


(* get the examinee record from disc *)
PROCEDURE LOADEXAMINEE;
BEGIN
  RESET(FILEEXAMINEE,INFONAME);
  SEEK(FILEEXAMINEE,RECNUM);
  GET(FILEEXAMINEE);
  EXAMINEE :[illegible]LEEXAMINEE^;
  CLOSE(FILEEXAMINEE,LOCK);
END;  (* load examinee *)


(* update examinee record *)
PROCEDURE UPDATEEXAMINEE;
BEGIN
  IF NOT DEMOFLAG THEN
  BEGIN
    RESET(FILEEXAMINEE,INFONAME);
    SEEK(FILEEXAMINEE,RECNUM);
    FILEEXAMINEE^ := EXAMINEE;
    PUT(FILEEXAMINEE);
    CLOSE(FILEEXAMINEE,LOCK);
  END;
END;  (* updateexaminee *)


(* update examinee directory *)
PROCEDURE UPDATEINDEX;
BEGIN
  IF NOT DEMOFLAG THEN
  BEGIN
    RESET(EDIR,EINDEX);
    SEEK(EDIR,0);
    EDIR^ := DIR;
    PUT(EDIR);
    CLOSE(EDIR,LOCK);
  END;
END;  (* update index *)



(* update test scores *)
PROCEDURE UPDATERESULTS;
BEGIN
  IF NOT DEMOFLAG THEN
  BEGIN
    RESET(FILETESTS,RESULTS);
    SEEK(FILETESTS,RECNUM);
    FILETESTS^ := TESTS;
    PUT(FILETESTS);
    CLOSE(FILETESTS,LOCK);
  END;
END; (* update results *)
```

```
(* load old test scores *)
PROCEDURE LOADRESULTS;
BEGIN
  RESET(FILETESTS,RESULTS);
  SEEK(FILETESTS,RECNUM);
  GET(FILETESTS);
  TESTS := FILETESTS^;
  CLOSE(FILETESTS,LOCK);
END;   (* loadresults *)



(* load information table *)
PROCEDURE LOADINFO;
BEGIN
  RESET(INFOFILE,TABNAME);
  SEEK(INFOFILE,RECNUM);
  GET(INFOFILE);
  INFOTABLE := INFOFILE^;
  CLOSE(INFOFILE,LOCK);
END; (* load info *)


(* load the set-up parameters *)
PROCEDURE LOADPARAMS;
BEGIN
  RESET(FILESPARAMS,SETUPDATA);
  SEEK(FILESPARAMS,0);
  GET(FILESPARAMS);
  SPARAMS := FILESPARAMS^;
  CLOSE(FILESPARAMS,LOCK);
END;    (* loadparams *)
```

```
(***************************************************************************)
(*                                                                         *)
(*      Textfile : ADMIN.DIR/A.PROCT.TEXT       Volume : TFILES            *)
(*      Codefile : ADMIN.CODE ('Include' file)  Volume : CATDATA           *)
(*                                                                         *)
(***************************************************************************)
(*               DEC.  1, 1982                  NPROC                      *)
(***************************************************************************)
(*                                                                         *)
(* Variable parameter passed back to indicate the state of affairs.        *)
(*                                                                         *)
(* Exit codes : 1 = return to testing procedures                           *)
(*              2 = exit general instructions.                             *)
(*              3 = exit log-in / continue                                 *)
(*              4 = exit subtest instructions/samples & continue           *)
(*              5 = go on to next subtest / discontinue present            *)
(*              6 = log-out examinee / end all modules / go on to          *)
(*                  next examinee & session.                               *)
(*                                                                         *)
(***************************************************************************)
(*                                                                         *)
(* Called from : 1 = general instructions                                  *)
(*               2 = login                                                 *)
(*               3 = instructions/samples                                  *)
(*               4 = administering question                                *)
(*               5 = driver subtest stop                                   *)
(*                                                                         *)
(***************************************************************************)
(*                                                                         *)
(*  When ever the proctor is called, the contents of the screen is lost,   *)
(*  thus certain codes are passed around to tell if the proctor was called.*)
(*  Then steps can be taken to restore what was on the screen before the   *)
(*  proctor call, eg. looping the writes to screen until key pressed was   *)
(*  not the help.                                                          *)
(*                                                                         *)
(***************************************************************************)

SEGMENT PROCEDURE CALLPROCTOR(CALLEDFROM : INTEGER; VAR EXITCODE : INTEGER);


CONST NUMPROCTOROPTIONS = 6;

      (* ring bell x times to call proctor *)
      MAXBELL = 5;

      (* password 1 *)
      P1WORD = 'LJ;';

      (* password 2 *)
      P2WORD = 'L1';



VAR SELECT,
    OPTION : CHAR;
    CONVERT : PACKED ARRAY [1..NUMPROCTOROPTIONS] OF CHAR;
    OKOPTIONS : SETOFCHAR;

    X,Y,I,J,K : INTEGER;
    PFLUSH,
    PASSWORD : STRING;

    DONE : BOOLEAN;


    PROCEDURE INVOPTION(OPT : INTEGER; INV : BOOLEAN);
    BEGIN
      IF INV THEN
        INVERSE
      ELSE
        NORMALSCR;

      CASE OPT OF
```

```
        1 : WRITE('RETURN TO TESTING PROCEDURES');
        2 : WRITE('EXIT GENERAL INSTRUCTIONS');
        3 : WRITE('EXIT LOG-IN / CONTINUE');
        4 : WRITE('EXIT SUBTEST INSTRUCTIONS/SAMPLES');
        5 : WRITE('GO ON TO NEXT SUBTEST');
        6 : WRITE('GO ON TO NEXT SESSION');
      end;

      IF INV THEN
        NORMALSCR
      ELSE
        INVERSE;
    END;

    PROCEDURE PMENU;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(13,1);
      WRITE('PROCTOR MENU');
      GOTOXY(1,4);
      WRITE('Select one of the following options by');
      GOTOXY(1,5);
      WRITE('entering its number, then pressing the');
      GOTOXY(1,6);
      WRITE('<YES> key.');

      J := 10;
      K := 1;

      (* display valid options *)
      FOR I := 1 to NUMPROCTOROPTIONS DO
      BEGIN
        IF CHR(I+48) IN OKOPTIONS THEN
        BEGIN
          GOTOXY(3,J);
          CASE I OF
            1 : WRITE(K,'. RETURN TO TESTING PROCEDURES');
            2 : WRITE(K,'. EXIT GENERAL INSTRUCTIONS');
            3 : WRITE(K,'. EXIT LOG-IN / CONTINUE');
            4 : WRITE(K,'. EXIT SUBTEST INSTRUCTIONS/SAMPLES');
            5 : WRITE(K,'. GO ON TO NEXT SUBTEST');
            6 : WRITE(K,'. GO ON TO NEXT SESSION');
          end;
          J := J + 1;

          (* save association between option and menu *)
          CONVERT[K] := CHR(I+48);
          K := K + 1;
        END;
      END;

      GOTOXY(0,22);
      WRITE(' Enter choice #, then press <YES> : ');

    END;  (* pmenu *)


BEGIN  (* callproctor *)

  (* save total number of proctor calls *)
  EXAMINEE.NUMPROC := EXAMINEE.NUMPROC + 1;

  (* save proctor calls during this test *)
  TESTS.STPROCTCALLS := TESTS.STPROCTCALLS + 1;


  IF GRAFIX THEN
  BEGIN
    PAGE(OUTPUT);
    TEXTON;
  END;
```

```
 IF NOT FORTYCOLUMN THEN
   TEXT40MODE;

 IF NOT REVERSEVIDEO THEN
   INVERSE;

 FOR I := 1 TO MAXBELL DO
   SQUAWK;    (* notify proctor *)

 (* set valid proctor menu options *)
 CASE CALLEDFROM OF

   (* called from general instructions *)
   1 : OKOPTIONS := ['1','2','6'];

   (* called from login *)
   2 : OKOPTIONS := ['1','3','6'];

   (* called from subtest instructions/samples *)
   3 : OKOPTIONS := ['1','4'..'6'];

   (* called from administer question *)
   4 : OKOPTIONS := ['1','5'..'6'];

   (* called from driver stop flag *)
   5 : OKOPTIONS := ['1'];
 END;  (* case *)

 (* tell examinee to get proctor *)
 PAGE(OUTPUT);
 GOTOXY(8,10);
 WRITE('Please call the proctor. ');

 (* get password 1 *)
 REPEAT
   GOTOXY(34,10);
   if ghost then
     pflush := p1word
   else
     READLN(KEYBOARD,PFLUSH);
   IF PFLUSH <> P1WORD THEN
 UNTIL PFLUSH = P1WORD;

 (* get password 2 *)
 PAGE(OUTPUT);
 GOTOXY(8,10);
 WRITE(' Password : ');

 REPEAT
   GOTOXY(20,10);
   if ghost then
     password := p2word
   else
     READLN(KEYBOARD,PASSWORD);
   IF PASSWORD <> P2WORD THEN
     FOR I := 1 TO MAXBELL DO
       SQUAWK;
 UNTIL PASSWORD = P2WORD;


 (**** display proctor menu ****)
 PMENU;


 (* select an option *)
 if ghost then
 begin
   select := ghostdigit;
   if select > chr(k+47) then
      select := '1';
    if select = '0' then
      select := '1';
 end
```

58

```
else
SELECT := GETCHAR(['1'..CHR(K+47)],true,TRUE,true);
OPTION := CONVERT[(ORD(SELECT)-48)];

Y := ORD(SELECT) - 39;
X := ORD(OPTION) - 48;


GOTOXY(6,Y);
INVOPTION(X,FALSE);


DONE := FALSE;
REPEAT

  GOTOXY(0,22);
  WRITE(' Enter choice #, then press <YES> : ');

  (* select an option *)
  if ghost then
    select := chr(yeskey)
  else
    SELECT := GETCHAR(['1'..CHR(K+47),CHR(YESKEY)],true,TRUE,true);

  IF SELECT <> CHR(YESKEY) THEN
  BEGIN
    GOTOXY(6,Y);
    INVOPTION(X,TRUE);
    OPTION := CONVERT[(ORD(SELECT)-48)];
    Y := ORD(SELECT) - 39;
    X := ORD(OPTION) - 48;
    GOTOXY(6,Y);
    INVOPTION(X,FALSE);
  END
  ELSE
    DONE := TRUE;


UNTIL DONE;

(* pass back flow code *)
EXITCODE := ORD(OPTION) - 48;

(* set global flow code *)
IF EXITCODE >= 5 THEN
   FLOWCODE := EXITCODE;

(* perform action based on option *)
CASE OPTION OF
  '1' : ;
  '2' : ;
  '3' : ;
  '4' : ;
  '5' : ;
  '6' : (* leaving session *)
            (* must save point *)
            (* of departure    *)
            BEGIN
              IF ONUM = CURRTLENGTH THEN
              BEGIN
                (* leaving session, done with subtest *)
                EXAMINEE.LASTTEST := TINDEX;

                (* check if it was the last test in the series *)
                IF TINDEX < GMAXSUBTEST THEN
                  IF SPARAMS.SUBORDER[TINDEX+1] < 0 THEN
                    (* if so, set done flag for examinee *)
                    EXAMINEE.LASTTEST := 128;
              END
              ELSE

                (* leaving session, not done with subtest *)
                EXAMINEE.LASTTEST := TINDEX - 1;
            END;
  END;

  IF NOT FORTYCOLUMN THEN
    TEXT80MODE;

  IF NOT REVERSEVIDEO THEN
    NORMALSCR
  ELSE
    INVERSE;

END;  (* callproctor *)
```

```
(**************************************************************************)
(*                                                                        *)
(*      Textfile : ADMIN.DIR/A.CF.TEXT          Volume : TFILES           *)
(*      Codefile : ADMIN.CODE ('Include' file)  Volume : CATDATA          *)
(*                                                                        *)
(**************************************************************************)
(* File last modified :  Jan 26, 1983                       NPROC         *)
(**************************************************************************)

(* This procedure gives a computer familarization session to the examinee *)
(* by requesting that a certain key is pressed.  Each key on the keypad   *)
(* is demonstrated and the examinee must press it within so many trys or  *)
(* else the proctor gets called automatically.                            *)

SEGMENT PROCEDURE COMPUTERFAMILARIZATION;

CONST MAXGIERROR = 3;    (* maximum amount of errors you can make *)


VAR REVERSE,          (* holds char value to write reverse vidio *)
    ANSWER,           (* chars that examinee presses *)
    KEYCHAR : CHAR;

    OK,                   (* loop flags *)
    DONE    : BOOLEAN;

    GCODE,      (* controls flow in this  module *)
    I,
    ASCIIVALUE,
    GIERROR : INTEGER;  (* number of errors made *)

    (* multiple chars we want examinee to type in *)
    KEYSTR2 : PACKED ARRAY[0..1] OF CHAR;
    KEYSTR6 : PACKED ARRAY[0..5] OF CHAR;
    KEYSTR3 : PACKED ARRAY[0..2] OF CHAR;
    KEYSTR9 : PACKED ARRAY[0..8] OF CHAR;


    (* draw the numeric keypad to screen *)
    PROCEDURE DRAWKEYBOARD;
    BEGIN
      PAGE(OUTPUT);
      WRITELN('  &&&&     &&&&&&&&&&&&&&&&&&&&&&&&');
      WRITELN(' |HELP|    | &&&     &&&     &&&  |');
      WRITELN('  &&&&     || 7 |   | 8 |   | 9 | |');
      WRITELN('           | &&&     &&&     &&&  |');
      WRITELN('           | &&&     &&&     &&&  |');
      WRITELN('           || 4 |   | 5 |   | 6 | |');
      WRITELN('           | &&&     &&&     &&&  |');
      WRITELN('           | &&&     &&&     &&&  |');
      WRITELN('           || 1 |   | 2 |   | 3 | |');
      WRITELN(' keypad &&>| &&&     &&&     &&&  |');
      WRITELN('           | &&&&&&&              |');
      WRITELN('           ||   0   |             |');
      WRITELN('           | &&&&&&&              |');
      WRITELN('           |  &&&&      &&&&&&&   |');
      WRITELN('           | | NO |    |  YES  |  |');
      WRITELN('           |  &&&&      &&&&&&&   |');
      WRITELN('           &&&&&&&&&&&&&&&&&&&&&&&&');
    END;  (* drawkeyboard *)


    (* draw inverse or normal key on keypad *)
    PROCEDURE INVERSEMODE(ON : BOOLEAN; KEY : CHAR);

        PROCEDURE INVWRITE(KEYSTR : STRING);
        BEGIN
          IF ON THEN
            REVERSE := CHR(17)
          ELSE
            REVERSE := CHR(18);
      WRITE(REVERSE,KEYSTR)
    END;  (* invwrite *)

BEGIN  (* inversemode *)
  ASCIIVALUE := ORD(KEY);
  CASE ASCIIVALUE OF
    48        : BEGIN
                  GOTOXY(15,11);
                  INVWRITE('  0  ');
                END;
    49        : BEGIN
                  GOTOXY(14,8);
                  INVWRITE(' 1 ');
                END;
    50        : BEGIN
                  GOTOXY(22,8);
                  INVWRITE(' 2 ');
                END;
    51        : BEGIN
                  GOTOXY(30,8);
                  INVWRITE(' 3 ');
                END;
    52        : BEGIN
                  GOTOXY(14,5);
                  INVWRITE(' 4 ');
                END;
    53        : BEGIN
                  GOTOXY(22,5);
                  INVWRITE(' 5 ');
                END;
    54        : BEGIN
                  GOTOXY(30,5);
                  INVWRITE(' 6 ');
                END;
    55        : BEGIN
                  GOTOXY(14,2);
                  INVWRITE(' 7 ');
                END;
    56        : BEGIN
                  GOTOXY(22,2);
                  INVWRITE(' 8 ');
                END;
    57        : BEGIN
                  GOTOXY(30,2);
                  INVWRITE(' 9 ');
                END;
    65        : BEGIN
                  GOTOXY(3,3);
                  INVWRITE(' A ');
                  GOTOXY(16,15);
                  INVWRITE('A');
                END;
    66        : BEGIN
                  GOTOXY(10,3);
                  INVWRITE(' B ');
                  GOTOXY(16,15);
                  INVWRITE('B');
                END;
    67        : BEGIN
                  GOTOXY(17,3);
                  INVWRITE(' C ');
                  GOTOXY(16,15);
                  INVWRITE('C');
                END;
    68        : BEGIN
                  GOTOXY(24,3);
                  INVWRITE(' D ');
                  GOTOXY(16,15);
                  INVWRITE('D');
                END;
    69        : BEGIN
                  GOTOXY(31,3);
```

62

```
                 INVWRITE(' E ');
                 GOTOXY(16,15);
                 INVWRITE('E');
               END;
    NOKEY    : BEGIN
                 GOTOXY(15,14);
                 INVWRITE(' NO ');
               END;
    YESKEY   : BEGIN
                 GOTOXY(27,14);
                 INVWRITE(' YES ');
               END;
    ERASEKEY : BEGIN
                 GOTOXY(26,11);
                 INVWRITE(' ERASE ');
               END;
    HELPKEY  : BEGIN
                 GOTOXY(1,1);
                 INVWRITE(' HELP ');
               END;
  END;  (* case *)
END;  (* inversemode *)




(* make examinee find 7 on keypad *)
PROCEDURE FIND7;
BEGIN
  GIERROR := 0;   (* initialize count of errors *)
  OK := FALSE;

  (* loop until enters in right key *)
  WHILE (NOT OK) DO
  BEGIN

    WRITELN(' Find the ''7'' key at the upper left');
    WRITE(' and press it.');

    (* make the key stand out *)
    INVERSEMODE(TRUE,'7');
    if ghost then
    begin
       if gierror >= maxgierror then
         keychar := '7'
       else
         keychar := ghostdigit
    end
    else
      KEYCHAR:=GETCHAR([CHR(ESC),CHR(HELPKEY)]+DIGITS,TRUE,FALSE,TRUE);
    INVERSEMODE(FALSE,'7');
    DELAY(1);
    IF KEYCHAR = CHR(HELPKEY) THEN
      CALLPROCT(1,GCODE);

    IF GCODE <> 1 THEN EXIT(COMPUTERFAMILARIZATION);

    IF KEYCHAR = CHR(HELPKEY) THEN
    BEGIN
      DRAWKEYBOARD;
      WRITELN;
    END
    ELSE
    BEGIN
      BLANKLINES(17,7,17);
      IF KEYCHAR <> '7' THEN
      BEGIN
        IF ORD(KEYCHAR)  = ESC THEN
        BEGIN
          SKIP := TRUE;
          EXIT(COMPUTERFAMILARIZATION);
        END
        ELSE
          BEGIN
            SQUAWK;
            WRITELN(' You pressed the ''',KEYCHAR,''' key instead');
            WRITELN(' of the ''7'' key.  Lets try it again.');
            WRITELN;
            GIERROR := GIERROR + 1;
            IF GIERROR = MAXGIERROR THEN
            BEGIN
              CALLPROCT(1,GCODE);
              IF GCODE <> 1 THEN
                EXIT(COMPUTERFAMILARIZATION);
              DRAWKEYBOARD;
              WRITELN;
            END;
          END;
      END
      ELSE
      BEGIN
        GOTOXY(0,19);
        OK := TRUE;
        WRITELN(' Good!');
        WRITELN;
      END;
    END;   (* if *)
  END;  (* while *)
END;  (* find7 *)


(* make examinee press 8 & 9 *)
PROCEDURE FIND89;
BEGIN
  GIERROR := 0;
  OK := FALSE;
  WHILE (NOT OK) DO
  BEGIN
    WRITE(' Press the 8 key and then the 9');
    INVERSEMODE(TRUE,'8');
    if ghost then
    begin
       if gierror >= maxgierror then
         keychar := '8'
       else
         keychar := ghostdigit
    end
    else
      KEYCHAR:=GETCHAR([CHR(HELPKEY)] + DIGITS,TRUE,FALSE,TRUE);
    INVERSEMODE(FALSE,'8');
    IF KEYCHAR = CHR(HELPKEY) THEN
    BEGIN
      CALLPROCT(1,GCODE);
      IF GCODE <> 1 THEN
        EXIT(COMPUTERFAMILARIZATION);
      DRAWKEYBOARD;
      BLANKLINES(19,4,19);
    END
    ELSE
    BEGIN
      KEYSTR2[0] := KEYCHAR;
      INVERSEMODE(TRUE,'9');
      if ghost then
      begin
        if gierror >= maxgierror then
          keychar := '9'
        else
          keychar := ghostdigit
      end
      else
        KEYCHAR:=GETCHAR([CHR(HELPKEY)] + DIGITS,TRUE,FALSE,TRUE);
      INVERSEMODE(FALSE,'9');
      DELAY(1);
      IF KEYCHAR = CHR(HELPKEY) THEN
      BEGIN
        CALLPROCT(1,GCODE);
        IF GCODE <> 1 THEN
          EXIT(COMPUTERFAMILARIZATION);
        DRAWKEYBOARD;
        BLANKLINES(19,4,19);
      END
      ELSE
      BEGIN
        KEYSTR2[1] := KEYCHAR;
        BLANKLINES(19,4,19);
        IF KEYSTR2 = '89' THEN
        BEGIN
          WRITELN(' Good!');
          OK := TRUE;
        END
        ELSE
        BEGIN
          SQUAWK;
          WRITELN(' You pressed ',KEYSTR2[0],' and ',
                   KEYSTR2[1],' instead of 8 ');
          WRITELN(' and 9.  Lets try it again.');
          GIERROR := GIERROR + 1;
          IF GIERROR = MAXGIERROR THEN
          BEGIN
            CALLPROCT(1,GCODE);
            IF GCODE <> 1 THEN
              EXIT(COMPUTERFAMILARIZATION);
            DRAWKEYBOARD;
            BLANKLINES(19,4,19);
          END;
        END;
        WRITELN;
      END;
    END;
  END; (* while *)
END;  (* find89 *)


(* examinee presses 1..6 *)
PROCEDURE FIND1TO6;
VAR SKIP,BADKEY : BOOLEAN;
BEGIN
  GIERROR := 0;
  OK := FALSE;
  WHILE (NOT OK) DO
  BEGIN
    WRITE(' Now type 123456 : ');
    I := 0;
    BADKEY := FALSE;
    SKIP := FALSE;
    KEYSTR6 := '      ';
    REPEAT
      if ghost then
      begin
        if gierror >= maxgierror then
          keychar := chr(ord(i+49))
        else
          keychar := ghostdigit
      end
      else
        KEYCHAR:=GETCHAR([CHR(HELPKEY)] + DIGITS,TRUE,TRUE,TRUE);
      IF KEYCHAR = CHR(HELPKEY) THEN
      BEGIN
        CALLPROCT(1,GCODE);
        IF GCODE <> 1 THEN
          EXIT(COMPUTERFAMILARIZATION);
        DRAWKEYBOARD;
        BLANKLINES(19,4,21);
        SKIP := TRUE;
      END
      ELSE
      BEGIN
        KEYSTR6[I] := KEYCHAR;
        IF ORD(KEYCHAR) <> (I+49) THEN
          BADKEY := TRUE;
        I := I + 1;
      END;
    UNTIL (I > 5) OR (BADKEY) OR (SKIP);
    IF NOT SKIP THEN
    BEGIN
      BLANKLINES(19,4,19);
      IF KEYSTR6 <> '123456' THEN
      BEGIN
        SQUAWK;
        WRITELN(' You typed ',KEYSTR6,' instead of 123456');
        WRITELN(' Lets try it again.');
        GIERROR := GIERROR + 1;
        IF GIERROR = MAXGIERROR THEN
        BEGIN
          CALLPROCT(1,GCODE);
          IF GCODE <> 1 THEN
            EXIT(COMPUTERFAMILARIZATION);
          DRAWKEYBOARD;
          BLANKLINES(19,4,21);
        END;
      END
      ELSE
      BEGIN
        WRITELN(' Excellent!');
        OK := TRUE;
      END;
      WRITELN;
    END;
  END;  (* while *)
END;  (* find1to6 *)


(* make examinee find 0 on keypad *)
PROCEDURE FIND0;
BEGIN
  GIERROR := 0;
  OK := FALSE;
  WHILE (NOT OK) DO
  BEGIN
    WRITE(' Press the 0 key.');
    INVERSEMODE(TRUE,'0');
    if ghost then
    begin
      if gierror >= maxgierror then
        keychar := '0'
      else
        keychar := ghostdigit;
    end
    else
      KEYCHAR:=GETCHAR([CHR(HELPKEY)] + DIGITS,TRUE,FALSE,TRUE);
    INVERSEMODE(FALSE,'0');
    IF KEYCHAR = CHR(HELPKEY) THEN
    BEGIN
      CALLPROCT(1,GCODE);
      IF GCODE <> 1 THEN
        EXIT(COMPUTERFAMILARIZATION);
      DRAWKEYBOARD;
      BLANKLINES(18,4,17);
    END
    ELSE
    BEGIN
      BLANKLINES(18,4,18);
      IF KEYCHAR <> '0' THEN
      BEGIN
        SQUAWK;
        WRITELN(' You pressed the ''',KEYCHAR,''' key instead of');
```

```
            WRITELN(' ''0'' key.  Lets try it again.');
            GIERROR := GIERROR + 1;
            IF GIERROR = MAXGIERROR THEN
            BEGIN
              CALLPROCT(1,GCODE);
              IF GCODE <> 1 THEN
                EXIT(COMPUTERFAMILARIZATION);
              DRAWKEYBOARD;
              BLANKLINES(18,4,17);
            END;
          END
          ELSE
          BEGIN
            WRITELN(' Good!');
            OK := TRUE;
          END;
          WRITELN;
        END;
      END;  (* while *)
    END;  (* find0 *)


    (* find the yes key *)
    PROCEDURE FINDYES;
    BEGIN
      OK := FALSE;
      GIERROR := 0;
      WHILE (NOT OK) DO
      BEGIN
        WRITELN(' Find the <YES> key at bottom right.');
        WRITELN(' Do you see it?  Press <YES>');
        INVERSEMODE(TRUE,CHR(YESKEY));
        if ghost then
          keychar := chr(yeskey)
        else
          KEYCHAR:=GETCHAR([CHR(HELPKEY),CHR(YESKEY),CHR(NOKEY)]+DIGITS,
                          TRUE,FALSE,TRUE);
        INVERSEMODE(FALSE,CHR(YESKEY));
        IF KEYCHAR = CHR(HELPKEY) THEN
        BEGIN
          CALLPROCT(1,GCODE);
          IF GCODE <> 1 THEN
            EXIT(COMPUTERFAMILARIZATION);
          DRAWKEYBOARD;
          BLANKLINES(18,4,18);
        END
        ELSE
        BEGIN
          BLANKLINES(18,5,19);
          IF ORD(KEYCHAR) <> YESKEY THEN
          BEGIN
            SQUAWK;
            WRITELN(' Thats not correct. Please try again.');
            GIERROR := GIERROR + 1;
            IF GIERROR = MAXGIERROR THEN
            BEGIN
              CALLPROCT(1,GCODE);
              IF GCODE <> 1 THEN
                EXIT(COMPUTERFAMILARIZATION);
              DRAWKEYBOARD;
              BLANKLINES(18,4,18);
            END;
          END
          ELSE
            ok := true;

          (*
          BEGIN
            INVERSEMODE(TRUE,CHR(YESKEY));
            INVERSEMODE(FALSE,CHR(NOKEY));
            GOTOXY(0,19);
            WRITELN(' Good. Use the <YES> key to answer');
            WRITELN(' ''yes'' to a question from the');
        WRITE(' computer.');
        OK := TRUE;
        PSTALL(1,GCODE);
        INVERSEMODE(FALSE,CHR(YESKEY));
        IF ABS(GCODE) <> 1 THEN
          EXIT(COMPUTERFAMILARIZATION);
        IF GCODE = -1 THEN
        BEGIN
          DRAWKEYBOARD;
          GCODE := 1;
        END;
      END;
      *)

    END;
  END;  (* while *)
END;  (* find yes *)


(* find the no key *)
PROCEDURE FINDNO;
BEGIN
  OK := FALSE;
  GIERROR := 0;
  BLANKLINES(18,6,19);
  WHILE (NOT OK) DO
  BEGIN
    WRITELN(' Use the <NO> key at the bottom left');
    WRITELN(' to answer ''no'' to a question from');
    WRITELN(' the computer. Find the <NO> key and');
    WRITE(' press it.');
    INVERSEMODE(TRUE,CHR(NOKEY));
    if ghost then
      keychar := chr(nokey)
    else
      KEYCHAR:=GETCHAR([CHR(HELPKEY),CHR(YESKEY),CHR(NOKEY)]+DIGITS,
                    TRUE,FALSE,TRUE);
    INVERSEMODE(FALSE,CHR(NOKEY));
    IF KEYCHAR = CHR(HELPKEY) THEN
    BEGIN
      CALLPROCT(1,GCODE);
      IF GCODE <> 1 THEN
        EXIT(COMPUTERFAMILARIZATION);
      DRAWKEYBOARD;
      BLANKLINES(18,6,18);
    END
    ELSE
    BEGIN
      BLANKLINES(18,6,18);
      IF ORD(KEYCHAR) <> NOKEY THEN
      BEGIN
        SQUAWK;
        WRITELN(' You did not press the <NO> key');
        GIERROR := GIERROR + 1;
        IF GIERROR = MAXGIERROR THEN
        BEGIN
          CALLPROCT(1,GCODE);
          IF GCODE <> 1 THEN
            EXIT(COMPUTERFAMILARIZATION);
          DRAWKEYBOARD;
          BLANKLINES(18,6,18);
        END;
      END
      ELSE
      BEGIN
        OK := TRUE;
      END;
    END;
  END;  (* while *)
END;  (* find no  *)
```

```
PROCEDURE USERERASE(STR, NEW_STR : STRING);
BEGIN
  GOTOXY(0,21);
  WRITE(' TYPE ',STR[1],STR[2],STR[3]);

  GOTOXY(6,22);
  FOR I := 1 TO 3 DO
    BEGIN
      if ghost then
        keychar := str[i]
      else
        KEYCHAR := GETCHAR([STR[I]],TRUE,TRUE,TRUE);
    END;

  BLANKLINES(21,1,21);
  WRITE(' Now erase the ',STR[3],' by pressing <ERASE>');
  INVERSEMODE(TRUE,CHR(ERASEKEY));
  GOTOXY(9,22);
  if ghost then
    keychar := chr(erasekey)
  else
    KEYCHAR := GETCHAR([CHR(ERASEKEY)],TRUE,FALSE,TRUE);
  INVERSEMODE(FALSE,CHR(ERASEKEY));
  GOTOXY(8,22);
  WRITE(' ');
  BLANKLINES(21,1,18);

  WRITELN(' Good.  To correct a mistake, erase it');
  WRITELN(' and type the correct key. Now replace ');
  WRITELN(' ',STR[2],' with a ',NEW_STR[1],' by pressing <ERASE> then ',
          NEW_STR[1]);
  INVERSEMODE(TRUE,CHR(ERASEKEY));
  GOTOXY(8,22);
  if ghost then
    keychar := chr(erasekey)
  else
    KEYCHAR:= GETCHAR([CHR(ERASEKEY)],TRUE,FALSE,TRUE);
  INVERSEMODE(FALSE,CHR(ERASEKEY));
  GOTOXY(7,22);
  WRITE(' ');

  INVERSEMODE(TRUE,NEW_STR[1]);
  GOTOXY(7,22);
  if ghost then
    keychar := newstr[1]
  else
    KEYCHAR:= GETCHAR([NEW_STR[1]],TRUE,TRUE,TRUE);
  INVERSEMODE(FALSE,NEW_STR[1]);
  BLANKLINES(18,6,21);

END;     (* Usererase *)



PROCEDURE ALPHA_BOARD;
BEGIN
  PAGE(OUTPUT);
  GOTOXY(0,2);
  WRITELN('   &&&    &&&    &&&    &&&    &&& ');
  WRITELN('  | A |  | B |  | C |  | D |  | E |');
  WRITELN('   &&&    &&&    &&&    &&&    &&& ');

END;     (* ALPHA_BOARD *)


PROCEDURE ALPHA_GEN_INSTR;
BEGIN
  GOTOXY(0,8);
  WRITELN('  Just to the left of the number keys');
  WRITELN('  are five keys labeled A, B, C, D, E.');
  WRITELN;
  WRITELN('  You will use these keys to enter');
  WRITELN('  your answers on certain tests.');

END;     (* ALPHA_GEN_INSTR *)



PROCEDURE ALPHA_KEY_PRESSES;
VAR CH, KEY_CH : CHAR;
BEGIN
  FOR CH := 'A' TO 'E' DO
    BEGIN
      BLANKLINES(15,1,15);
      WRITELN('  Now, find the ',CH,' key and press it!');
      INVERSEMODE(TRUE,CH);
      if ghost then
        keycnar := ch
      else
        KEYCHAR := GETCHAR([CH],TRUE,FALSE,TRUE);
      INVERSEMODE(FALSE,CH);
    END;

  PAGE(OUTPUT);

END;     (* ALPHA_KEY_PRESSES *)



PROCEDURE ALPHA_FAMILARIZATION;
VAR
  G_ARRAY : PACKED ARRAY[0..0] OF 5..6;
BEGIN
  (* Turn Cursor Off *)
  G_ARRAY[0] := 6;
  UNITWRITE(1,G_ARRAY,1,,12);

  ALPHA_BOARD;
  ALPHA_GEN_INSTR;
  ALPHA_KEY_PRESSES;

  (* Turn Cursor On *)
  G_ARRAY[0] := 5;
  UNITWRITE(1,G_ARRAY,1,,12);

END;     (* ALPHA_FAMILARIZATION *)



(* ask if wants instructions again  *)
PROCEDURE PROMPT;
BEGIN
  PAGE(OUTPUT);
  GOTOXY(0,10);
  WRITELN(' Do you want to practice using the');
  WRITE(' computer again? ');
  if ghost then
    keychar := chr(nokey)
  else
    KEYCHAR:=GETCHAR(KEYPAD,TRUE,TRUE,TRUE);
  BLANKLINES(18,2,18);
  IF ORD(KEYCHAR) = NOKEY THEN
  BEGIN
    CALLPROCT(1,GCODE);
    IF GCODE <> 1 THEN
      EXIT(COMPUTERFAMILARIZATION);
  END;
END;  (* prompt *)
```

```
    (* introduce the help key *)
    PROCEDURE LOCHELP;
    var stallchar : char;
        I,J       : integer;
    BEGIN
      DRAWKEYBOARD;
      BLANKLINES(17,7,18);
      WRITELN(' Finally, find the <HELP> key at the');
      WRITELN(' top of the keyboard.');
      WRITELN(' When you need help, press this key');
      WRITELN(' to call the proctor. ');
      WRITELN;
      WRITE(' Press the <HELP> key now. ');

      repeat
        INVERSEMODE(TRUE,CHR(HELPKEY));
        if ghost then
          stallchar := chr(helpkey)
        else
          stallchar :=
            getchar([chr(helpkey),chr(yeskey),chr(esc)],true,true,true);
        INVERSEMODE(FALSE,CHR(HELPKEY));
        blanklines(17,7,20);
        if stallchar <> chr(helpkey) then
            write(' Try again...  Press the <HELP> key. ')
            else
          begin
            write('             Very good!');
            DELAY(4);
          end;
      until stallchar = chr(helpkey);

      INVERSEMODE(FALSE,CHR(YESKEY));
      IF ABS(GCODE) <> 1 THEN
        EXIT(COMPUTERFAMILARIZATION);
      GCODE := 1;
    END;    (* lochelp *)


BEGIN (* computer familarization *)
  GCODE := 1;
  DONE := FALSE;
  REPEAT
    DRAWKEYBOARD;
    WRITELN(' Welcome.  Here are some instructions');
    WRITELN(' on how to use the test computer.');
    WRITELN(' First, notice the numeric keypad ');
    WRITELN(' on the far right of the keyboard.');
    WRITELN;
    FIND7;
    FIND89;
    FIND1TO6;
    FIND0;
    FINDYES;
    FINDNO;

    (*
    USERERASE('123','7');
    USERERASE('789','4');
    USERERASE('987','2');
    *)

    ALPHA_FAMILARIZATION;
    LOCHELP;

    (*
    PAGE(OUTPUT);
    GOTOXY(0,13);
    WRITE(' Do you understand everything so far? ');
    if ghost then
      keychar := chr(yeskey
    else
      KEYCHAR:=GETCHAR([CHR(NOKEY),CHR(YESKEY)],TRUE,FALSE,TRUE);
    IF ORD(KEYCHAR) = NOKEY THEN
      PROMPT
    ELSE
      DONE := TRUE;
    *)

    done := true;
  UNTIL DONE;
END;    (* computer familarization *)
```

```
(*****************************************************************************)
(*                                                                           *)
(*      Textfile : ADMIN.DIR/A.LOGIN.TEXT         Volume : TFILES            *)
(*      Codefile : ADMIN.CODE ('Include' file)    Volume : CATDATA           *)
(*                                                                           *)
(*****************************************************************************)
(*      File Last Modified  :  Oct   7,1983                    NPROC         *)
(*****************************************************************************)


(* get personal data and info from examinee *)
(* set up variables to handle returning or  *)
(* new examinees.                           *)
SEGMENT PROCEDURE LOGIN;
VAR LOGTRY,
    LCODE : INTEGER;   (* controls flow in this module *)

    LOGINOK,STAYOK : BOOLEAN;



    (* get directory index for an id number *)
    (* returns nil if no such id exists.    *)
    FUNCTION DIRINDEXNUM(IDNUM : IDTYPE) : INTEGER;
    VAR I : INTEGER;
        DONE : BOOLEAN;
    BEGIN
      LOADINDEX;
      I := 0;
      DONE := FALSE;
      DIRINDEXNUM := NIL;
      REPEAT
        IF (DIR[I].ID = IDNUM) AND
           (NOT (DIR[I].UNUSED)) THEN
        BEGIN
          DONE := TRUE;
          DIRINDEXNUM := I;
        END;
        I := I + 1;
      UNTIL (I > MAXEXAMINEE) OR (DONE);
    END;  (* dirindexnum *)


    PROCEDURE READGTHETA;
                              (* read as string => integer => real *)
    VAR
    REALSTR:STRING;
    NUM,DEC,DIG,I,SIZE:INTEGER;
    NEG:BOOLEAN;

    BEGIN

    NEG:=FALSE;
    READLN(REALSTR);
    IF POS('-',REALSTR) > 0 THEN
    BEGIN
      DELETE(REALSTR,1,1);
      NEG:=TRUE;
    END;

    DEC:=POS('.',REALSTR);
    DELETE(REALSTR,DEC,1);
    SIZE:=LENGTH(REALSTR);
    DIG:=0;
    FOR I:=1 TO SIZE
    DO BEGIN
         NUM:=ORD(REALSTR[I]) - 48;
         DIG:=NUM +(DIG * 10);
       END;

       GTHETA:=DIG;
       DEC:=SIZE -(DEC-1);
```

```
   FOR I:=1 TO DEC
   DO GTHETA:=GTHETA/10;
   IF NEG THEN GTHETA:= (-1 * GTHETA);
END;  (* READGTHETA *)


PROCEDURE GHOSTMENU;

VAR
CHOICE:INTEGER;

BEGIN
  TEXT80MODE;
  INVERSE;
  WRITE(CHR(28));
  GOTOXY(12,4);
  WRITE('GHOST OPTION  MENU');
  GOTOXY(8,6);
  WRITE('1.    RANDOM NUMBER GENERATOR GHOST');
  GOTOXY(8,8);
  WRITE('2.    SINGLE ABILITY LEVEL SIMULATION GHOST');
  GOTOXY(8,10);
  WRITE('3.    AUTOMATIC LOOP OF ABILITY LEVELS GHOST');
  GOTOXY(8,12);
  WRITE('ENTER CHOICE # :');
  READLN(CHOICE);
  TEXT40MODE;
  GHOSTFLOW:=CHOICE;
  IF GHOSTFLOW = SINGLE THEN
  BEGIN
    WRITE(CHR(28));
    GOTOXY(1,12);
    WRITE('ENTER A VALUE FOR THETA :');
    READGTHETA;
  END;
END; (* GHOSTMENU *)

(* get examinee id number *)
PROCEDURE ENTERID;
VAR OKID : BOOLEAN;
    IDSTR : STRING;
    K : INTEGER;
BEGIN
  IDSTR := '          ';
  OKID := FALSE;
  REPEAT
    PAGE(OUTPUT);
    FILLCHAR(LINEBUF[0],9,' ');
    GOTOXY(1,10);
    WRITE('Press <YES> when you are done entering');
    GOTOXY(1,11);
    WRITE('your number.');
    GOTOXY(1,5);
    WRITE('Enter your');
    GOTOXY(26,6);
    WRITE('---------');
    GOTOXY(1,6);
    WRITE('Social Security Number : ');
    (* read in 9 digit social security number *)
    if ghost then
    begin
      IF GHOSTFLOW = AUTO
      THEN
      BEGIN
        WRITE('GHOSTAUTO');
        IF ABLOOP = MAXGTHETA
        THEN ABLOOP:= MINGTHETA
        ELSE
        ABLOOP:= SUCC(ABLOOP);
      END
      ELSE IF GHOSTFLOW = SINGLE
           THEN WRITE('GHOSTSING')
```

```
         ELSE
         write('GHOST');
     delay(2);
     exit(enterid);
    end
    else
     PFILLBUF(9,DIGITS + ['R','A','C','G','H','S','T','D','E','M','O','Y'],
                          TRUE,2,LCODE);
    IF LCODE = 1 THEN
    BEGIN
     MOVELEFT(LINEBUF[0],EXAMINEE.ID[0],9);
     IF EXAMINEE.ID = '         ' THEN
      SQUAWK
     ELSE
      OKID := TRUE;
    END
    ELSE
    BEGIN
     IF LCODE <> -1 THEN
     BEGIN
      (* examinee leaves before log in of id # then session is invalid *)
      FLOWCODE := 2;
      DEMOFLAG := TRUE;
      EXIT(LOGIN);
     END;
     LCODE := 1;
    END;
   UNTIL OKID;
   FILLCHAR(LINEBUF[0],9,' ');

   FOR K := 1 TO 9 DO
    IDSTR[K] := EXAMINEE.ID[K-1];

   (* set flag if demo *)
   IF POS('DEMO',IDSTR) <> 0  THEN
    DEMOFLAG := TRUE
   ELSE
    DEMOFLAG := FALSE;

   IF POS('GHOST',IDSTR) <> 0 THEN
   BEGIN
    GHOST := TRUE;
    GHOSTMENU;
    SKIPFAM:= TRUE;
   END
   ELSE
    GHOST := FALSE;

   IF POS('TRACE',IDSTR) <> 0 THEN
    TRACE := TRUE
   ELSE
    TRACE := FALSE;

  IF POS('STAY',IDSTR) <> 0 THEN (* Debug routine, program remains in row *)
   STAY := TRUE                 (* Uncomment routines in A.QUEST to use *)
   ELSE
    STAY := FALSE;
  END;  (* enter id *)


  (* give instructions *)
  PROCEDURE LOGINSTRUCTIONS;
  BEGIN
   (* loop until dont want instructions again *)
   REPEAT

    (* set flow to normal *)
    LCODE := 1;

    (* log-in instructions *)
    PAGE(OUTPUT);
    GOTOXY(10,1);
```

75

```
        WRITE('LOG-IN INSTRUCTIONS');
        GOTOXY(0,3);
        WRITELN(' The computer will ask you for your');
        WRITELN(' Social Security Number.');
        WRITELN;
        WRITELN(' At that time type in your');
        WRITELN(' Social Security Number and press');
        WRITELN(' the <YES> key after the Social');
        WRITELN(' Security Number is entered.');
        WRITELN;
        (* WRITELN(' If you make a mistake typing,');
        WRITELN(' use the <ERASE> key to correct');
        WRITELN(' the error.');
        WRITELN;*)
        WRITELN(' Remember to press the <YES> key after');
        WRITELN(' entering your Social Security Number.');
        WRITELN(' Call the proctor if you have any');
        WRITELN(' questions.');
        WRITELN;
        WRITE(' To continue,');
        PSTALL(2,LCODE);
        IF ABS(LCODE) <> 1 THEN
        BEGIN
          FLOWCODE := ABS(LCODE);
          EXIT(LOGIN);
        END;
      UNTIL LCODE = 1;
    END;   (* Log instructions *)

BEGIN   (* log in *)

  LOGINSTRUCTIONS;
  LOGTRY := 0;

  REPEAT
    LOGINOK := TRUE;
    LCODE := 1;
    (* initialize line buffer *)
    FILLCHAR(LINEBUF[0],79,' ');

    ENTERID;   (* get id number *)

    GOTOXY(13,22);
    WRITE('Please wait ');

    IF STAY THEN
    BEGIN
      STAYOK:=FALSE;
      REPEAT
        WRITE(CHR(28));
        GOTOXY(0,10);
        WRITE('ENTER THETA ROW VALUE (1 TO 36) :');
        READLN(LEVEL);
        IF (LEVEL > 36) OR (LEVEL < 1) THEN
          WRITE(CHR(7))
        ELSE
          STAYOK :=TRUE;
      UNTIL STAYOK;
    END;

    (* this is a demo *)
    IF (DEMOFLAG) or (ghost) OR (TRACE) OR (STAY)   THEN
    BEGIN
      DEMOFLAG := TRUE;
      ERECNUM := 0;
      TINDEX := 0;
      NEWEXAMINEE := TRUE;
      TNUM := 0;
      EXAMINEE.TESTLENGTH := SPARAMS.SUBLENGTH;
      EXIT(LOGIN);
    END;

    ERECNUM := DIRINDEXNUM(EXAMINEE.ID);
```

```
IF (ERECNUM >= 0)  THEN    (* examinee has been entered into system *)

BEGIN
    LOADEXAMINEE(ERECNUM);

    IF (EXAMINEE.LASTTEST < GMAXSUBTEST) and (not ghost) THEN
    BEGIN
      NEWEXAMINEE := FALSE;

      (* setup examinee to resume old test *)

      (* set index into configuration array where last left off *)
      TINDEX := EXAMINEE.LASTTEST;

      (* set next place to store results record *)
      TNUM := ERECNUM * GMAXSUBTEST + TINDEX;

      (* load in previous results of unfinished test *)
      LOADRESULTS(TNUM);

    END
    ELSE
      IF (EXAMINEE.LASTTEST = GMAXSUBTEST)  THEN
      BEGIN
        NEWEXAMINEE := TRUE;
        (* save the examinee's configuration *)
        WITH EXAMINEE DO
        BEGIN
          TESTORDER := SPARAMS.SUBORDER;
          SUBSTOP := SPARAMS.SUBSTOP;
          CKERROR := SPARAMS.CKERROR;
          TESTLENGTH := SPARAMS.SUBLENGTH;
          STRATEGY := SPARAMS.SUBSTRAT;
          DATE := SYSTEMDATE;
        END;

        (* set the logical record number to store results on disk *)
        TNUM := ERECNUM * GMAXSUBTEST;

        (* set the index into configuration arrays *)
        TINDEX := 0;

        (* update examinee record on disk *)
        UPDATEEXAMINEE(ERECNUM);

        (* update examinee directory *)
        UPDATEINDEX;
      END
      ELSE
      BEGIN
        FLOWCODE := 6;     (* examinee done, dont give session *)
        DEMOFLAG := TRUE;
        GOTOXY(13,22);
        WRITE('             ');
        GOTOXY(0,13);
        WRITELN;
        WRITELN;
        WRITELN(' This examinee done with session already.');
        WRITELN;
        WRITELN;
        SQUAWK;
        STALL;
      END;
  END
  ELSE
  BEGIN
    GOTOXY(13,22);
    WRITE('             ');
    GOTOXY(0,13);
    WRITELN;
    WRITELN;
    WRITELN(' Access for ',EXAMINEE.ID,' not verified.');
    WRITELN;
```

77

```
        WRITELN;
        SQUAWK;
        STALL;
        LOGINOK := FALSE;
        LOGTRY := LOGTRY + 1;
        LCODE := 1;
        IF LOGTRY > 3 THEN
          CALLPROCTOR(2,LCODE);
        IF LCODE >= 3 THEN
        BEGIN
          IF LCODE = 3 THEN
            FLOWCODE := 6;
          EXIT(LOGIN);
        END;
      END;UNTIL LOGINOK;
END; (* log-in *)
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : ADMIN.DIR/A.GI.TEXT          Volume : TFILES               *)
(*      Codefile : ADMIN.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(* File last modified : Jun 15, 1983                             NPROC        *)
(******************************************************************************)

    (* give test taking information *)
    SEGMENT PROCEDURE GENERALINSTRUCTIONS;
    VAR ANSWER,
        KEYCHAR : CHAR;
        GCODE : INTEGER;
        DONE : BOOLEAN;


    (* ask if wants instructions again  *)
    PROCEDURE PROMPT;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(0,10);
      WRITELN(' Do you want to go through the');
      WRITE(' instructions again? ');
      if ghost then
        keychar := chr(nokey)
      else
        KEYCHAR:=GETCHAR(KEYPAD,TRUE,TRUE,TRUE);
      BLANKLINES(18,2,18);
      IF ORD(KEYCHAR) = NOKEY THEN
      BEGIN
        CALLPROCT(1,GCODE);
        IF GCODE <> 1 THEN
          EXIT(GENERALINSTRUCTIONS);
      END;
    END;  (* prompt *)


    (* test taking info *)
    PROCEDURE TINFO1;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(9,1);
      WRITELN(' HOW TO TAKE THE TESTS');
      WRITELN;
      WRITELN(' Most of the questions on the tests are');
      WRITELN(' multiple choice, that is... there will');
      WRITELN(' be a question followed by a list of ');
      WRITELN(' choices.  For example....');
      WRITELN;
      WRITE(' How many eggs are there in a dozen?');
      GOTOXY(15,11);
      WRITE(' A. 10');
      GOTOXY(15,13);
      WRITE(' B. 16');
      GOTOXY(15,15);
      WRITE(' C. 12');
      GOTOXY(15,17);
      WRITE(' D. 24');
      GOTOXY(0,19);
      REPEAT
        WRITE(' Your answer : ');
        if ghost then
          answer := 'C'
        else
          ANSWER:=GETCHAR(['A'..'D'],TRUE,TRUE,TRUE);
        WRITELN;
        WRITELN;
        WRITE(' Is ''',ANSWER,''' the answer you want to give? ');
        if ghost then
          keychar := chr(yeskey)
        else
```

```
          KEYCHAR:=GETCHAR([CHR(YESKEY),CHR(NOKEY)],TRUE,FALSE,TRUE);
        IF ORD(KEYCHAR) = NOKEY THEN
          BLANKLINES(19,3,19);
      UNTIL ORD(KEYCHAR) = YESKEY;
      BLANKLINES(21,1,21);
      IF ANSWER = 'C' THEN
        WRITELN(' Thats right.')
      ELSE
        WRITELN(' The correct answur is C.');
      WRITELN;
      PSTALL(1,GCODE);
      IF ABS(GCODE) <> 1 THEN
        EXIT(GENERALINSTRUCTIONS);
      GCODE := 1;
    END;  (* tinfol *)


    (* more test taking info *)
    PROCEDURE TINFO2;
    BEGIN
      PAGE(OUTPUT);
      WRITELN;
      WRITELN(' Here is another example......');
      WRITELN;
      WRITELN;
      WRITELN;
      WRITELN;
      WRITELN;
      WRITE(' What is two plus two?');
      GOTOXY(15,10);
      WRITE(' A. Eight');
      GOTOXY(15,12);
      WRITE(' B. Four');
      GOTOXY(15,14);
      WRITE(' C. Eleven');
      GOTOXY(15,16);
      WRITE(' D. Twenty');
      GOTOXY(0,21);
      REPEAT
        WRITE(' Your answer : ');
        if ghost then
          answer := 'A'
        else
          ANSWER:=GETCHAR(['A'..'D'],TRUE,TRUE,TRUE);
        WRITELN;
        WRITELN;
        WRITE(' Is ''',ANSWER,''' the answer you want to give? ');
        if ghost then
          keychar := chr(yeskey)
        else
          KEYCHAR := GETCHAR([CHR(YESKEY),CHR(NOKEY)],TRUE,FALSE,TRUE);
        IF ORD(KEYCHAR) = NOKEY THEN
          BLANKLINES(21,3,21);
      UNTIL ORD(KEYCHAR) = YESKEY;
      BLANKLINES(21,3,21);
      IF ANSWER = 'B' THEN
        WRITELN(' Thats right.')
      ELSE
        WRITELN(' The correct answer is B.');
      WRITELN;
      PSTALL(1,GCODE);
      IF ABS(GCODE) <> 1 THEN
        EXIT(GENERALINSTRUCTIONS);
      GCODE := 1;
    END;  (* tinfo2 *)


BEGIN (* general instructions *)
  GCODE := 1;
  DONE := FALSE;
  REPEAT
    TINFO1;
    TINFO2;
    PAGE(OUTPUT);
    GOTOXY(0,13);
    WRITE(' Do you understand everything so far? ');
    if ghost then
      keychar := chr(yeskey)
    else
      KEYCHAR:=GETCHAR([CHR(NOKEY),CHR(YESKEY)],TRUE,FALSE,TRUE);
    IF ORD(KEYCHAR) = NOKEY THEN
      PROMPT
    ELSE
      DONE := TRUE;
    IF DONE THEN
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(0,9);
      WRITELN(' This is the end of your computer');
      WRITELN(' introduction. If you still are not');
      WRITELN(' comfortable with the keyboard or the');
      WRITELN(' instructions, please see the proctor');
      WRITELN(' for assistance.  Thank you.');
      WRITELN;
      WRITE(' To continue,');
      PSTALL(1,GCODE);
      IF ABS(GCODE) <> 1 THEN
        EXIT(GENERALINSTRUCTIONS);
      GCODE := 1;
      PAGE(OUTPUT);
      GOTOXY(0,6);
      WRITELN(' You should have scratch paper and a');
      WRITELN(' pencil for any figuring you need to');
      WRITELN(' do. Return the scratch paper when');
      WRITELN(' you finish the test.');
      WRITELN;
      WRITELN(' If another pencil or more paper is');
      WRITELN(' needed during the test, use the');
      WRITELN(' <HELP> key to call the proctor.');
      WRITELN;
      PSTALL(1,GCODE);
      IF ABS(GCODE) <> 1 THEN
        EXIT(GENERALINSTRUCTIONS);
    END;
  UNTIL DONE;
END;    (* general instructions *)
```

```
(**************************************************************************)
(*                                                                        *)
(*      Textfile : ADMIN.DIR/A.INITE.TEXT       Volume : TFILES           *)
(*      Codefile : ADMIN.CODE ('Include' file)  Volume : CATDATA          *)
(*                                                                        *)
(**************************************************************************)
(*               DEC.  1, 1982                  NPROC                     *)
(**************************************************************************)

(* This procedure initializes the examinees test result record and *)
(* takes into account if he is a new examinee or a returning       *)
(* examinee.                                                       *)

SEGMENT PROCEDURE INITEXAMINEE;
BEGIN


  (* initialize pool of previously used questions *)
  FOR QNUM := 0 TO QUESTIONS DO
    USEDQ[QNUM] := NIL;

  (* turn prefetching off for the first question either for returning to a *)
  (* subtest or starting a the beginning of a new subtest.                 *)
  FIRSTQUESTION := TRUE;

  SAMPLEQUESTION := FALSE;

  (* set flag for timed tests, time not expired yet *)
  TIMEOUT := FALSE;

  (* new examinee is first time they've logged in *)
  IF NEWEXAMINEE THEN
  BEGIN
    INITSCORES := TRUE;

    (* set all parameters to today's set up *)
    CURRSTRAT := SPARAMS.SUBSTRAT[TINDEX];
    CURRTLENGTH := SPARAMS.SUBLENGTH[TINDEX];
    CURRSUBSTOP := SPARAMS.SUBSTOP[TINDEX];
    CURRTEST := SPARAMS.SUBORDER[TINDEX];
  END
  ELSE
  BEGIN   (* examinee previously logged in, returning to complete tests *)

    (* reset pool of used questions that examinee has taken already *)
    IF NOT INITSCORES THEN
    BEGIN
      QNUM := -1;
      REPEAT
        QNUM := QNUM + 1;
        USEDQ[QNUM] := TESTS.ITEMINFO[QNUM].ITEMNUM;
      UNTIL (USEDQ[QNUM] < 0) OR (QNUM >= QUESTIONS);
    END;

    (* reset examinee's last ability and variance *)
    CURRABILITY := TESTS.ESTABILITY;
    CURRVARIANCE := TESTS.VARIANCE;

    (* reset elaspped time if timed subtest *)
    ELASPEDTIME := EXAMINEE.PREVTIMELASTTEST;


    (* reset all parameters set up for this examinee *)
    CURRSTRAT := EXAMINEE.STRATEGY[TINDEX];
    CURRTLENGTH := EXAMINEE.TESTLENGTH[TINDEX];
    CURRSUBSTOP := EXAMINEE.SUBSTOP[TINDEX];
    CURRTEST := EXAMINEE.TESTORDER[TINDEX];

    (* if test item count is zero, examinee is returning to the  *)
    (* beginning of a subtest and must initialize the test score *)
    IF TESTS.NUMITEMS <= 0 THEN
      INITSCORES := TRUE;
  END;

  (* if examinee returns to middle of subtest, don't initialize *)
  (* his test results record.                                   *)
  IF INITSCORES THEN
  BEGIN
    CURRABILITY := 0;    (* initialize examinee ability *)

    (* initialize examinee variance *)
    CURRVARIANCE := 1.0;

    (* initialize stopwatch for timed tests *)
    ELASPEDTIME := 0;


    (* initialize #items taken, #correct *)
    TESTS.NUMCORR := 0;
    TESTS.NUMITEMS := 0;
    TESTS.STPROCTCALLS := 0;

    (* mark all item codes nil <unused> *)
    FOR QNUM := 0 TO QUESTIONS DO
      TESTS.ITEMINFO[QNUM].ITEMNUM := NIL;

    (* initilize array index for current question *)
    QNUM := 0;

  END;
END;  (* initexaminee *)
```

```
(*****************************************************************************)
(*                                                                           *)
(*      Textfile : ADMIN.DIR/A.LOADT.TEXT       Volume : TFILES              *)
(*      Codefile : ADMIN.CODE ('Include' file)  Volume : CATDATA             *)
(*                                                                           *)
(*****************************************************************************)
(* File last modified :  OCT 18, 1983                     NPROC              *)
(*****************************************************************************)

(*  loads  directory of test , loads appropriate data structure for *)
(* strategy if any, gives instructions and samples for subtest.     *)

SEGMENT PROCEDURE LOADTEST;

CONST MAXSQERROR = 3; (* maximum # errors in samples before calling proctor *)

VAR QCOUNT,      (* # of questions on multiple question screen *)
    I,
    HASHLOC,     (* record # of data, block/byte ptrs *)
    BLOCK,       (* which block text starts *)
    BLOCKPTR,    (* which byte in block text starts *)
    SCODE,       (* flow control in this module *)
    SECS,        (* seconds for timed tests *)
    SQERROR   : INTEGER;  (* # of key errors in samples questions *)
    TIMECODE,
    TEXTCODE,    (* specifies screen format *)
    RCHAR     : CHAR;



    (* answer the sample questions *)
    PROCEDURE ANSWERSAMPLE;
    VAR CHRANS : CHAR;
        QACOUNT,
        INTANS : INTEGER;
        S7ANS  : SEVENTYPE;
        OK     : BOOLEAN;
    BEGIN

      (* answer type *)
      CASE ITEMINFO.ATYPE OF

        (* multiple choice *)
        CHARVALUE : BEGIN

                      (* get answer *)
                      CHRANS := GETCHRANSWER(3,SCODE);

                      (* scode <> 1, proctor was called *)
                      IF SCODE <> 1 THEN
                        EXIT(ANSWERSAMPLE);

                      (* sample feedback *)
                      IF CHRANS = ITEMINFO.ANSWER THEN
                      BEGIN
                        BLANKLINES(23,1,23);
                        WRITE('Thats correct! ');
                        STALL;
                      END
                      ELSE
                      BEGIN
                        (* count # of wrong key ins *)
                        SQERROR := 0;
                        REPEAT
                          BLANKLINES(21,3,21);
                          WRITELN('The correct answer is ',
                                  ITEMINFO.ANSWER,' not ',CHRANS);
                          WRITELN;
                          WRITE('Type ',ITEMINFO.ANSWER,
                                ' on your keypad : ');
                          if ghost then
```

```
                    chrans := iteminfo.answer
                  else
                    CHRANS :=
                     GETCHAR([ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER],
                             TRUE,TRUE,TRUE);
                  IF CHRANS <> ITEMINFO.ANSWER THEN
                  BEGIN
                    SQERROR := SQERROR + 1;
                    IF SQERROR = MAXSQERROR THEN
                    BEGIN

                      (* call proctor if refuses to press right key *)
                      CALLPROCT(3,SCODE);
                      IF SCODE <> 1 THEN
                        EXIT(ANSWERSAMPLES);

                      (* restore cscreen to what it was *)
                      DECODEPRINT(BLOCK,BLOCKPTR);
                    END;
                  END;
                UNTIL (CHRANS = ITEMINFO.ANSWER);
              END;
            END;

(* integer value as answer *)
INTVALUE  : BEGIN

              (* get an integer *)
              INTANS := GETINTANSWER(3,SCODE);
              IF SCODE <> 1 THEN
                EXIT(ANSWERSAMPLE);
              IF INTANS = ITEMINFO.INTANSWER THEN
              BEGIN
                BLANKLINES(23,1,23);
                WRITE('Thats correct! ');
                STALL;
              END
              ELSE
              BEGIN
                SQERROR := 0;
                REPEAT
                  BLANKLINES(21,3,21);
                  WRITELN('The correct answer is ',
                           ITEMINFO.INTANSWER,' not ',
                           INTANS);
                  WRITELN;
                  WRITE('Enter the correct answer : ');
                  if ghost then
                    intans := iteminfo.intanswer
                  else
                    INTANS := GETINTSTR;
                  IF INTANS <> ITEMINFO.INTANSWER THEN
                  BEGIN
                    SQERROR := SQERROR + 1;
                    IF SQERROR = MAXSQERROR THEN
                    BEGIN
                      CALLPROCT(3,SCODE);
                      IF SCODE <> 1 THEN
                        EXIT(ANSWERSAMPLE);
                      DECODEPRINT(BLOCK,BLOCKPTR);
                    END;
                  END;
                UNTIL INTANS = ITEMINFO.INTANSWER;
              END;
            END;

(* get an array of 7 chars *)
SEVENCHR  : BEGIN
              OCOUNT := ITEMINFO.ANSWERCOUNT;
              GETSEVENANSWERS(1,OCOUNT,3,BLOCK,BLOCKPTR,S7ANS,SCODE,
                               QACOUNT);
              IF SCODE <> 1 THEN
                EXIT(ANSWERSAMPLE);
```

```
                  BLANKLINES(21,3,20);
                  WRITE('Your  answers  were : ');
                  FOR I := 1 TO QCOUNT DO
                    WRITE('(',I,') ',S7ANS[I],'  ');
                  WRITELN;
                  WRITE('Correct answers are : ');
                  FOR I := 1 TO QCOUNT DO
                    WRITE('(',I,') ',ITEMINFO.CHRANSWER[I],'  ');
                  WRITELN;
                  WRITELN;
                  STALL;
                END;


  END;  (* case *)
  SCODE := 1;
END;  (* answer samples *)


(* answer the sample questions grafix version *)
PROCEDURE GANSWERSAMPLE;
VAR CHRANS : CHAR;
    QACOUNT,
    W,
    INTANS : INTEGER;
    S7ANS  : SEVENTYPE;
    OK     : BOOLEAN;
BEGIN

  (* answer type *)
  CASE ITEMINFO.ATYPE OF

    (* multiple choice *)
    CHARVALUE : BEGIN

                  (* get answer *)
                  CHRANS := GGETCHRANSWER(3,SCODE);

                  (* scode <> 1, proctor was called *)
                  IF SCODE <> 1 THEN
                    EXIT(GANSWERSAMPLE);

                  (* sample feedback *)
                  IF CHRANS = ITEMINFO.ANSWER THEN
                  BEGIN
                    GBLANKLINES(23,1,23);
                    GWRITESTR('Thats correct! ');
                    GSTALL;
                  END
                  ELSE
                  BEGIN
                    (* count # of wrong key ins *)
                    SQERROR := 0;
                    REPEAT
                      GBLANKLINES(21,3,21);
                      GWRITESTR('The correct answer is ');
                      GWRITECHR(ITEMINFO.ANSWER);
                      GWRITESTR(' not ');
                      GWRITECHR(CHRANS);
                      GWRITELN;
                      GWRITELN;
                      GWRITESTR('Type ');
                      GWRITECHR(ITEMINFO.ANSWER);
                      GWRITESTR(' on your keypad : ');
                      if ghost then
                        chrans := iteminfo.answer
                      else
                        CHRANS :=
                         GGETCHAR([ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER],
                                 TRUE,TRUE,TRUE);
                      IF CHRANS <> ITEMINFO.ANSWER THEN
                      BEGIN
                        SQERROR := SQERROR + 1;
                    IF SQERROR = MAXSQERROR THEN
                    BEGIN

                      (* call proctor if refuses to press right key *)
                      CALLPROCT(3,SCODE);
                      IF SCODE <> 1 THEN
                        EXIT(ANSWERSAMPLES);

                      (* restore cscreen to what it was *)
                      GDECODEPRINT(CURRTEST,I);
                    END;
                  END;
                UNTIL (CHRANS = ITEMINFO.ANSWER);
              END;
            END;

(* integer value as answer *)
INTVALUE  : BEGIN

              (* get an integer *)
              INTANS := GGETINTANSWER(3,SCODE);
              IF SCODE <> 1 THEN
                EXIT(GANSWERSAMPLE);
              IF INTANS = ITEMINFO.INTANSWER THEN
              BEGIN
                GBLANKLINES(23,1,23);
                GWRITESTR('Thats correct! ');
                GSTALL;
              END
              ELSE
              BEGIN
                SQERROR := 0;
                REPEAT
                  GBLANKLINES(21,3,21);
                  GWRITESTR('The correct answer is ');
                  GWRITEINT(ITEMINFO.INTANSWER);
                  GWRITESTR(' not ');
                  GWRITEINT(INTANS);
                  GWRITELN;
                  GWRITELN;
                  GWRITESTR('Enter the correct answer : ');
                  if ghost then
                    intans := iteminfo.intanswer
                  else
                    INTANS := GGETINTSTR;
                  IF INTANS <> ITEMINFO.INTANSWER THEN
                  BEGIN
                    SQERROR := SQERROR + 1;
                    IF SQERROR = MAXSQERROR THEN
                    BEGIN
                      CALLPROCT(3,SCODE);
                      IF SCODE <> 1 THEN
                        EXIT(ANSWERSAMPLE);
                      GDECODEPRINT(CURRTEST,I);
                    END;
                  END;
                UNTIL INTANS = ITEMINFO.INTANSWER;
              END;
            END;

(* get an array of 7 chars *)
SEVENCHR  : BEGIN
              QCOUNT := ITEMINFO.ANSWERCOUNT;
              GGETSEVENANSWERS(1,QCOUNT,3,CURRTEST,I,S7ANS,SCODE,
                                QACOUNT);
              IF SCODE <> 1 THEN
                EXIT(GANSWERSAMPLE);
              GBLANKLINES(21,3,20);
              GWRITESTR('Your  answers  were : ');
              FOR W := 1 TO QCOUNT DO
              BEGIN
                GWRITECHR('(');
                GWRITEINT(W);
```

```
                        GWRITESTR(') ');
                        GWRITECHR(S7ANS[W]);
                        GWRITESTR('  ');
                      END;
                      GWRITELN;
                      GWRITESTR('Correct answers are : ');
                      FOR W := 1 TO QCOUNT DO
                      BEGIN
                        GWRITECHR('(');
                        GWRITEINT(W);
                        GWRITESTR(') ');
                        GWRITECHR(ITEMINFO.CHRANSWER[W]);
                        GWRITESTR('  ');
                      END;
                      GWRITELN;
                      GWRITELN;
                      GSTALL;
                    END;


      END;  (* case *)
      SCODE := 1;
    END;  (* ganswer samples *)



BEGIN   (* loadtest *)


  RESET(FILEDIRECTORY,INDEXNAME);
  SEEK(FILEDIRECTORY,CURRTEST);
  GET(FILEDIRECTORY);


  (* load question index *)
  DIRECTORY := FILEDIRECTORY^;

  CLOSE(FILEDIRECTORY,LOCK);


  (* configure questions according to strategy *)
  CASE CURRSTRAT OF
    NONE        : ;
    TIMED       : BEGIN
                    (* get maximum time allowed in seconds *)
                    TIMECODE := DIRECTORY.TESTNAME[2];
                    MAXTIME := (ORD(TIMECODE) - 48) * 60;
                    TIMECODE := DIRECTORY.TESTNAME[3];
                    MAXTIME := MAXTIME + ((ORD(TIMECODE) - 48) * 10);
                    TIMECODE := DIRECTORY.TESTNAME[4];
                    MAXTIME := MAXTIME + (ORD(TIMECODE) - 48);
                  END;

    (* load information table *)
   B102222,
   B54321,
   B108642   : LOADINFO(CURRTEST + MAXSUBTESTS + 1);
  END;

  (* set default screen format 40 columns inverse *)
  REVERSEVIDEO := TRUE;
  FORTYCOLUMN := TRUE;

  (* check if any other type format desired *)
  TEXTCODE := DIRECTORY.TESTNAME[1];
  CASE TEXTCODE OF
    '*' : FORTYCOLUMN := FALSE;
    '@' : BEGIN
            FORTYCOLUMN := FALSE;
            REVERSEVIDEO := FALSE;
          END;
    '?' : REVERSEVIDEO := FALSE;
  END;

  IF FORTYCOLUMN THEN
    TEXT40MODE
  ELSE
    TEXT80MODE;

  IF REVERSEVIDEO THEN
    INVERSE
  ELSE
    NORMALSCR;

  (* loop instructions and samples until examinee comfortable *)
  REPEAT

    (* loop instructions *)
    REPEAT

      (* set flow to normal *)
      SCODE := 1;

      (* display subtest instructions *)
      IF DIRECTORY.ITEMCODE[0] >= 0 THEN
      BEGIN

        (* get record # to load ptrs to get text *)
        HASH_LOC := HASH(0);
        RESET(FILEITEMINFO,DATANAME);
        SEEK(FILEITEMINFO,HASH_LOC);
        GET(FILEITEMINFO);
        ITEMINFO := FILEITEMINFO^;
        CLOSE(FILEITEMINFO,LOCK);

        (* load ptrs for text *)
        BLOCK := ITEMINFO.ITEMBLOCK;
        BLOCKPTR := ITEMINFO.ITEMPTR;

        (* display the text *)
        DECODEPRINT(BLOCK,BLOCKPTR);

        (* allow examinee to call proctor if wants *)
        GOTOXY(0,21);
        PSTALL(3,SCODE);
        IF ABS(SCODE) > 4 THEN
          (* does not want to continue subtest or session *)
          EXIT(LOADTEST);
      END;
    UNTIL (SCODE = 1)        (* normal *)
          OR (SCODE = -4);   (* exit instructions *)

    (* reset back to normal *)
    SCODE := 1;

    (* give subtest samples *)
    SAMPLEQUESTION := TRUE;
    I := 1;
    WHILE (I <= 5) DO
    BEGIN
      IF DIRECTORY.ITEMCODE[I] > 0 THEN
      BEGIN

        (* get record # of data for this sample *)
        HASH_LOC := HASH(I);
        RESET(FILEITEMINFO,DATANAME);
        SEEK(FILEITEMINFO,HASH_LOC);
        GET(FILEITEMINFO);
        ITEMINFO := FILEITEMINFO^;
        CLOSE(FILEITEMINFO,LOCK);

        (* get text pointers for sample *)
        BLOCK := ITEMINFO.ITEMBLOCK;
        BLOCKPTR := ITEMINFO.ITEMPTR;
        (* give the sample until examinee answers or skips *)
        REPEAT

          (* reset normal flow *)
          SCODE := 1;

          (* display the sample *)
          IF ITEMINFO.GRAPHICS THEN
          BEGIN
            GRAFIX := TRUE;
            GDECODEPRINT(CURRTEST,I);
            PAGE(OUTPUT);
            GANSWERSAMPLE;
            GRAFIX := FALSE;
            TEXTON;
          END
          ELSE
          BEGIN
            DECODEPRINT(BLOCK,BLOCKPTR);
            ANSWERSAMPLE;
          END;

          IF ABS(SCODE) > 4 THEN
            (* wants to leave subtest or session *)
            EXIT(LOADTEST);

          IF ABS(SCODE) = 4 THEN
          (* wants to exit rest of samples *)
          BEGIN
            SCODE := 1;
            I := 5;
          END;
        UNTIL SCODE = 1;
      END;
      I := I + 1;
    END;

    (* ask if ready to begin test *)
    PAGE(OUTPUT);
    IF FORTYCOLUMN THEN
      GOTOXY(1,10)
    ELSE
      GOTOXY(21,10);
    WRITE('Are you ready to begin the test? ');
    if ghost then
      rchar := chr(yeskey)
    else
      RCHAR := GETCHAR([CHR(YESKEY),CHR(NOKEY)],TRUE,FALSE,TRUE);
    IF RCHAR = CHR(NOKEY) THEN
    BEGIN
      PAGE(OUTPUT);
      IF FORTYCOLUMN THEN
        GOTOXY(1,10)
      ELSE
        GOTOXY(21,10);
      WRITELN('Do you want to go over the instructions');

      IF NOT FORTYCOLUMN THEN
        GOTOXY(21,11);
      WRITE(' and samples again? ');
      IF GETCHAR([CHR(YESKEY),CHR(NOKEY)],TRUE,FALSE,TRUE)=CHR(NOKEY) THEN
        (* call proctor if not ready and doesnt want more instructions *)
        CALLPROCT(3,SCODE);
      IF SCODE <> 1 THEN
        EXIT(LOADTEST);
    END;
  UNTIL (RCHAR = CHR(YESKEY));
  SAMPLEQUESTION := FALSE;
END;  (* loadtest *)
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : ADMIN.DIR/A.QUEST.TEXT       Volume : TFILES               *)
(*      Codefile : ADMIN.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(*      File last modified :  Aug  5, 1983          NPRDC                     *)
(******************************************************************************)

(* This procedure searches through an appropriate strategy data structure *)
(* and selects a question to give the examinee based on certain data.  It *)
(* also checks to see if the question has been used before and gives one  *)
(* which hasn't been used.  It elicits an answer from the examinee and    *)
(* updates the record with his response.                                  *)

PROCEDURE ADMINISTERQUESTION;
VAR QCOUNT,               (* # of questions per item in special tests *)
    QSTCODE,              (* code # of question *)
    QSLOT,                (* slot in directory *)
    BLK,                  (* block where text starts *)
    BLKPTR,               (* byte in block where text starts *)
    I,
    INTANS,               (* integer answer to question *)
    DATASLOT,             (* record # where text ptrs are and data *)
    ACODE : INTEGER;      (* flow variable for this module *)

    CHRANS : CHAR;        (* multiple choice answer *)

    S7ANS  : SEVENTYPE;  (* seven char answer *)
    GATE,
    DONE : BOOLEAN;

    START,                (* time question appeared *)
    STOP : REAL;          (* time examinee answered question *)


    (* this function searches an array of used question *)
    (* codes and returns nil if the question was unused *)
    FUNCTION SEARCH(QCODE : INTEGER) : INTEGER;
    VAR FOUND : BOOLEAN;
        I : INTEGER;
    BEGIN
      IF QCODE < 0 THEN  (* invalid code #, set that it was used so wont use *)
        SEARCH := 1000   (* any positive number will do *)
      ELSE
      BEGIN
        FOUND := FALSE;
        I := 0;
        REPEAT
          IF USEDQ[I] = QCODE THEN
            FOUND := TRUE
          ELSE
            I := I + 1;
        UNTIL (FOUND) OR (I > QNUM) OR (I > QUESTIONS);
        IF FOUND THEN
          SEARCH := I
        ELSE
          SEARCH := NIL;
      END;
    END;  (* search *)

    PROCEDURE ERROR(IROW:INTEGER);
    VAR PASSWORD,PWORD:STRING;
        I:INTEGER;

    BEGIN
      PAGE(OUTPUT);
      WRITELN('INFOTABLE SELECT ERROR !!!!!!');
      WRITELN;
      WRITELN('NO MORE QUESTIONS AT THIS ABILITY LEVEL');
      WRITELN;
      WRITELN('INFOTABLE ROW ',IROW);
    WRITELN;
    STALL;
     (*
    WRITELN(SAVERC[1],' ',SAVERC[2]);
    WRITELN(SAVERC[3],' ',SAVERC[4]);
    *)
    FOR I:=1 TO 5 DO
    BEGIN
      WRITE(I,' ',INFOTABLE[I,IROW],' ');
      WRITE(I+5,' ',INFOTABLE[I+5,IROW],' ');
      WRITE(I+10,' ',INFOTABLE[I+10,IROW],' ');
      WRITE(I+15,' ',INFOTABLE[I+15,IROW] ;
      WRITELN;
    END;
    WRITELN;
     (*
    WRITELN('THE CONTENTS OF THE USEDQ ARRAY');
    FOR I:=0 TO 4 DO
    BEGIN
      WRITE(I,' ',USEDQ[I],' ');
      WRITE(I+5,' ',USEDQ[I+5],' ');
      WRITE(I+10,' ',USEDQ[I+10],' ');
      WRITE(I+15,' ',USEDQ[I+15],' ');
      WRITELN;
    END;
    *)
    PWORD:='LJ';
    WRITELN;
    WRITELN('PROCTOR : COPY DATA ON SCREEN ');
    WRITELN('THEN ENTER PASSWORD :');
    REPEAT
      READLN(PASSWORD);
    UNTIL PASSWORD = PWORD;
    WRITE(CHR(28));
  END;

  (************************************************************)
  (* select a question from the information table based on the *)
  (* examinees current ability level.                          *)
  (* takes six parameters, 1. ability level used to find the   *)
  (* appropriate row, and 5 steps which specify the number of  *)
  (* questions to randomly select from for the nth question    *)
  (* all questions after the fifth question take on the fifth  *)
  (* step.                                                     *)
  (************************************************************)

  FUNCTION INFOSELECT(ABILITY : REAL; S1,S2,S3,S4,S5 : INTEGER) : INTEGER;

  VAR QSELECT,
      GETMAXQ,
      USED,
      ROW,
      COUNT,
      COLUMN : INTEGER;
      T : REAL;
      OKQ : BOOLEAN;
      SAVECOL,
      POOL : ARRAY[1..10] OF INTEGER;
  BEGIN  (* infoselect *)
    (* row based on 36 levels ranging from -2.250 to +2.125 *)
    (* ability rises 8x's slower than table index           *)
    (* 19 is the offset into the table                      *)
    ROW := TRUNC((ABILITY * 8) + 19.5000000);
    IF STAY THEN
      ROW:=LEVEL;
    IF ROW > 36 THEN
      ROW := 36
    ELSE
      IF ROW < 1 THEN
        ROW := 1;
    if trace then
    begin
      gotoxy(0,0);
  blanklines(0,1,0);
  write('infotable row is ',row);
  readln;
end;

(* get the number of random  questions to select the next quest. from *)
IF FIRSTQUESTION THEN
BEGIN
  GETMAXQ := S1;
END
ELSE
BEGIN
  GETMAXQ := S5;  (* default *)
  CASE QNUM OF
    0 : GETMAXQ := S2;
    1 : GETMAXQ := S3;
    2 : GETMAXQ := S4;
  END;
END;

if trace then
begin
  gotoxy(0,0);
  blanklines(0,1,0);
  write('maximum # of random questions looked at ',getmaxq);
  readln;
end;

OKQ := FALSE;
REPEAT

  (* get the maximum # of questions not used already *)
  COLUMN := 1;
  COUNT := 0;
  REPEAT

    (* check if question previously used *)
    IF INFOTABLE[COLUMN,ROW] < 0 THEN (* infotable= -1 if the question *)
                                      (* is not available  *)
        USED := 1
      ELSE
        USED := SEARCH(INFOTABLE[COLUMN,ROW]); (*search returns -1 if *)
        (* the question # is not in the array of used questions       *)

    (* if question wasnt used save it *)
    IF USED < 0 THEN
    BEGIN
      COUNT := COUNT + 1;
      POOL[COUNT] := INFOTABLE[COLUMN,ROW];
      SAVECOL[COUNT] := COLUMN;
    END
    ELSE
      (*mark entry in table as used *)
      INFOTABLE[COLUMN,ROW] := -1 ;

    COLUMN := COLUMN + 1;
  UNTIL (COLUMN > 20) OR (COUNT >= GETMAXQ);

  IF (COLUMN > 20) AND (COUNT = 0) THEN
  BEGIN
    ERROR(ROW);
    ROW :=ROW + 1;
    IF ROW > 36 THEN ROW :=35;
    LEVEL:=ROW;
  END
  ELSE
  BEGIN
    (* randomly choose from available pool *)
    T := RANDOM;
    QSELECT := (TRUNC(T) MOD COUNT) + 1;
    INFOSELECT := POOL[QSELECT];
    OKQ := TRUE;
    COLUMN := SAVECOL[QSELECT];
```

```
      (* Save the row and column index for update of the infotable *)
      (* once it is known if examinee answered right or wrong      *)
      IF RIGHT THEN
      BEGIN
        SAVERC[1]:= ROW;
        SAVERC[2]:= COLUMN;
        RIGHT:= FALSE;
      END
      ELSE
      BEGIN
        SAVERC[3]:= ROW;
        SAVERC[4]:= COLUMN;
        RIGHT:= TRUE;
      END;

      if trace then
      begin
        gotoxy(0,0);
        blanklines(0,1,0);
        write('column  is ',column);
        readln;
        gotoxy(0,0);
        qselect := pool[qselect];
        write('question is ',directory.itemcode[qselect]);
        readln;
      end;

    END;
  UNTIL OKQ;
END;  (* infoselect *)


(**********************************************)
(* This function steps through randomly the   *)
(* test directory and selects the first item  *)
(* not previously given.  There is no strategy *)
(* involved.                                   *)
(**********************************************)

FUNCTION NOSTRATSELECT : INTEGER;
VAR USED,
    SINDEX : INTEGER;
    OKQ : BOOLEAN;
BEGIN
  SINDEX := QNUM + MAXSAMPLES + 1;
  OKQ := FALSE;
  REPEAT
    IF DIRECTORY.ITEMCODE[SINDEX] >= 0 THEN
    BEGIN
      USED := SEARCH(SINDEX);
      IF USED < 0 THEN
        OKQ := TRUE
    END;
    IF NOT OKQ THEN
      SINDEX := SINDEX + 1;
  UNTIL (OKQ) OR (SINDEX >= MAXITEMPOOL);
  IF OKQ THEN
    NOSTRATSELECT := SINDEX
  ELSE
  BEGIN
    PAGE(OUTPUT);
    WRITE('No STRATEGY QUESTION SELECT ERROR!!!!!');
    GOTOXY(0,10);
    STALL;
  END;
END;   (* nostratselect *)


(* answer the displayed question *)
PROCEDURE ANSWERQUESTION;
VAR QANSWERED : INTEGER;      (* number of questions answered *)
```

```
        SAVEITEM : BOOLEAN;
    BEGIN
      (**** get the answer and save the results ****)
      CASE ITEMINFO.ATYPE OF
        CHARVALUE : BEGIN
                      IF ITEMINFO.GRAPHICS THEN
                        CHRANS := GGETCHRANSWER(4,ACODE)
                      ELSE
                        CHRANS := GETCHRANSWER(4,ACODE);
                      IF ACODE = 1 THEN
                      BEGIN
                        SAVEITEM := TRUE;

                        (* update record *)
                        IF CURRSTRAT = TIMED THEN
                        BEGIN
                          STOP := TIME;
                          TESTS.ITEMINFO[QNUM].LATENCY := ABS(START-STOP);
                          FLASPEDTIME := ELASPEDTIME + (TRUNC(ABS(START-STOP)));
                          IF ELASPEDTIME >= MAXTIME THEN
                            SAVEITEM := FALSE;
                        END;
                        IF SAVEITEM THEN
                        BEGIN
                          TESTS.MOOSTTIME := ELASPEDTIME;
                          TESTS.ITEMINFO[QNUM].RTYPE := CHARVALUE;
                          TESTS.ITEMINFO[QNUM].RESPONSE := CHRANS;
                          TESTS.ITEMINFO[QNUM].ITEMNUM := QSTCODE;
                          TESTS.NUMITEMS := TESTS.NUMITEMS + 1;
                          IF CHRANS = ITEMINFO.ANSWER THEN
                          BEGIN
                            TESTS.NUMCORR := TESTS.NUMCORR + 1;
                            TESTS.ITEMINFO[QNUM].CORRECT := TRUE;
                            IF  (CURRSTRAT = B102222) OR
                                (CURRSTRAT = B54321)  OR
                                (CURRSTRAT = B108642) THEN
                            BEGIN
                              QPREFETCH := RITEPREFETCH;
                              CURRABILITY := RITEABILITY;
                              CURRVARIANCE := RITEVARIANCE;
                              RIGHT := TRUE;
                            END;
                          END
                          ELSE
                          BEGIN
                            TESTS.ITEMINFO[QNUM].CORRECT := FALSE;
                            IF  (CURRSTRAT = B102222) OR
                                (CURRSTRAT = B54321)  OR
                                (CURRSTRAT = B108642) THEN
                            BEGIN
                              QPREFETCH := WRONGPREFETCH;
                              CURRABILITY := WRONGABILITY;
                              CURRVARIANCE := WRONGVARIANCE;
                              RIGHT := FALSE;
                            END;
                          END;
                        END;
                      END;
                    END;

        INTVALUE    : BEGIN
                      IF ITEMINFO.GRAPHICS THEN
                        INTANS := GGETINTANSWER(4,ACODE)
                      ELSE
                        INTANS := GETINTANSWER(4,ACODE);
                      IF ACODE = 1 THEN
                      BEGIN
                        SAVEITEM := TRUE;
                        IF CURRSTRAT = TIMED THEN
                        BEGIN
                          STOP := TIME;
                          TESTS.ITEMINFO[QNUM].LATENCY := ABS(START-STOP);
                          ELASPEDTIME := ELASPEDTIME + (TRUNC(ABS(START-STOP)));
```

```
                  IF ELASPEDTIME >= MAXTIME THEN
                    SAVEITEM := FALSE;
                END;
                IF SAVEITEM THEN
                BEGIN
                  TESTS.MODSTTIME := ELASPEDTIME;
                  TESTS.ITEMINFO[QNUM].RTYPE := INTVALUE;
                  TESTS.ITEMINFO[QNUM].INTRESPONSE := INTANS;
                  TESTS.ITEMINFO[QNUM].ITEMNUM := QSTCODE;
                  TESTS.NUMITEMS := TESTS.NUMITEMS + 1;
                  IF INTANS = ITEMINFO.INTANSWER
                  THEN
                  BEGIN
                    TESTS.NUMCORR := TESTS.NUMCORR + 1;
                    TESTS.ITEMINFO[QNUM].CORRECT := TRUE;
                    IF (CURRSTRAT = B102222) OR
                       (CURRSTRAT = B54321)  OR
                       (CURRSTRAT = B108642) THEN
                    BEGIN
                      QPREFETCH := RITEPREFETCH;
                      CURRABILITY := RITEABILITY;
                      CURRVARIANCE := RITEVARIANCE;
                      RIGHT := TRUE;
                    END;
                  END
                  ELSE
                  BEGIN
                    TESTS.ITEMINFO[QNUM].CORRECT := FALSE;
                    IF (CURRSTRAT = B102222) OR
                       (CURRSTRAT = B54321)  OR
                       (CURRSTRAT = B108642) THEN
                    BEGIN
                      QPREFETCH := WRONGPREFETCH;
                      CURRABILITY := WRONGABILITY;
                      CURRVARIANCE := WRONGVARIANCE;
                      RIGHT := FALSE;
                    END;
                  END;
                END;
              END;
            END;

SEVENCHR  : BEGIN
              QCOUNT := ITEMINFO.ANSWERCOUNT;
              IF ITEMINFO.GRAPHICS THEN
                GETSEVENANSWERS(1,QCOUNT,4,CURRTEST,
                                QSTCODE,S7ANS,ACODE,
                                QANSWERED)
              ELSE
                GETSEVENANSWERS(1,QCOUNT,4,BLK,BLKPTR,S7ANS,ACODE,
                                QANSWERED);
              IF QANSWERED > 0 THEN
              BEGIN
                IF ACODE = 1 THEN
                BEGIN
                  IF CURRSTRAT = TIMED THEN
                  BEGIN
                    STOP := TIME;
                    ELASPEDTIME := ELASPEDTIME +
                                   (TRUNC(ABS(START-STOP)));
                    IF ELASPEDTIME >= MAXTIME THEN
                      TESTS.ITEMINFO[QNUM].LATENCY := ABS(START-STOP) -
                      (ELASPEDTIME - MAXTIME)
                    ELSE
                      TESTS.ITEMINFO[QNUM].LATENCY := ABS(START-STOP);
                  END;
                  TESTS.ITEMINFO[QNUM].RTYPE := SEVENCHR;
                  TESTS.ITEMINFO[QNUM].CHRRESPONSE := S7ANS;
                  TESTS.ITEMINFO[QNUM].ITEMNUM := QSTCODE;
                  TESTS.NUMITEMS := TESTS.NUMITEMS + QANSWERED;
                  TESTS.ITEMINFO[QNUM].ACOUNT := QANSWERED;
                  FOR I := 1 TO QANSWERED DO
                  BEGIN
              IF STANS[I] = ITEMINFO.CHRANSWER[I] THEN
              BEGIN
               TESTS.NUMCORR := TESTS.NUMCORR + 1;
               TESTS.ITEMINFO[QNUM].ACORRECT[I-1] := TRUE;
              END
              ELSE
               TESTS.ITEMINFO[QNUM].ACORRECT[I-1] := FALSE;
             END;
            END;
           END;
          END;
   END;  (* case *)
  END;   (* answer question *)


  (* precalculating to get next question *)
  PROCEDURE PRECALCULATE;
  BEGIN

   (* set current ability and variance for prefetch *)
   RITEABILITY := CURRABILITY;
   WRONGABILITY := CURRABILITY;
   RITEVARIANCE := CURRVARIANCE;
   WRONGVARIANCE := CURRVARIANCE;

   (* calculate abilities if answered wrong and right *)
   (* THIS ROUTINE IS SKIPPED IF THE STAY ROUTINE IS USED *)
   IF NOT STAY THEN
    UPDATEABILITY(TRUE);

   (* calculate next question if examinee answers current one correctly *)
   CASE CURRSTRAT OF
    B102222     : RITEPREFETCH := INFOSELECT(RITEABILITY,10,2,2,2,2);
    B54321      : RITEPREFETCH := INFOSELECT(RITEABILITY,5,4,3,2,1);
    B108642     : RITEPREFETCH := INFOSELECT(RITEABILITY 10,8,6,4,2);
   END;

   (* calculate next question if examinee answers current one wrong *)
   CASE CURRSTRAT OF
    B102222     : WRONGPREFETCH := INFOSELECT(WRONGABILITY,10,2,2,2,2);
    B54321      : WRONGPREFETCH := INFOSELECT(WRONGABILITY,5,4,3,2,1);
    B108642     : WRONGPREFETCH := INFOSELECT(WRONGABILITY,10 8,6,4,2);
   END;
  END;


BEGIN  (* administer question  *)

 CASE CURRSTRAT OF
  NONE,
  TIMED       : QSLOT := NOSTRATSELECT;
  B102222     : IF FIRSTQUESTION THEN
                BEGIN
                 QSLOT := INFOSELECT(CURRABILITY,10,2,2,2,2);
                 FIRSTQUESTION := FALSE;
                 RIGHT := TRUE;
                END
                ELSE
                 QSLOT := QPREFETCH;
  B54321      : IF FIRSTQUESTION THEN
                BEGIN
                 QSLOT := INFOSELECT(CURRABILITY,5,4,3,2,1);
                 FIRSTQUESTION := FALSE;
                 RIGHT := TRUE;
                END
                ELSE
                 QSLOT := QPREFETCH;
  B108642     : IF FIRSTQUESTION THEN
                BEGIN
                 QSLOT := INFOSELECT(CURRABILITY,10,8,6,4,2);
                 FIRSTQUESTION := FALSE;
```

```
                  RIGHT:= TRUE ;
                END
                ELSE
                  QSLOT := QPREFETCH;
END;

(*
if trace then
begin
  writeln;
  writeln;
  writeln('Question selected = ',qstcode);
  writeln;
  stall;
end;
*)

USEDQ[QNUM] := QSLOT;  (* keep track of used questions *)

(* Mark question as use in the infotable *)
IF RIGHT THEN
  INFOTABLE[SAVERC[2],SAVERC[1]]:= -1
ELSE
  INFOTABLE[SAVERC[4],SAVERC[3]]:= -1;
RIGHT := TRUE;

QSTCODE := DIRECTORY.ITEMCODE[QSLOT]; (* save code # *)

IF QSTCODE >= 0 THEN (* question exists *)

(***** show question *****)
BEGIN


  if trace then
  begin
    gotoxy(0,0);
    blanklines(0,1,0);
    write('item selected is ',qstcode);
    readln;
  end;

  DATASLOT := HASH(QSLOT);  (* find question data record # *)

  (* get the question data *)
  RESET(FILEITEMINFO,DATANAME);
  SEEK(FILEITEMINFO,DATASLOT);
  GET(FILEITEMINFO);
  ITEMINFO := FILEITEMINFO^;
  CLOSE(FILEITEMINFO,LOCK);

  (* get pointers to find text *)
  BLK := ITEMINFO.ITEMBLOCK;
  BLKPTR := ITEMINFO.ITEMPTR;

  DONE := FALSE;
  GATE := TRUE;
  REPEAT

    (* set flow to normal *)
    ACODE := 1;

    (* display the text *)
    IF ITEMINFO.GRAPHICS THEN
    BEGIN
      GRAFIX := TRUE;
      FILLPORT;
      GDECODEPRINT(CURRTEST,QSTCODE);
      PAGE(OUTPUT);
    END
    ELSE
    BEGIN
      DECODEPRINT(BLK,BLKPTR);  (* display the question *)
```

98

```
        UNITCLEAR(2);     (* flush the keyboard buffer *)
      END;

      (* prefetching of the next question while examines is reading the *)
      (* current question .                                             *)
      IF (CURRSTRAT = B102222) OR
         (CURRSTRAT = B54321)  OR
         (CURRSTRAT = B100642)  THEN
        PRECALCULATE;

      (* start the timer for timed tests *)
      IF (CURRSTRAT = TIMED) AND (GATE) THEN
      BEGIN
        GATE := FALSE;
        START := TIME;
      END;

      (* get the examinee response to question *)
      ANSWERQUESTION;

      TEXTON;
      GRAFIX := FALSE;

      IF ACODE = 1 THEN
        DONE := TRUE;


      (* if taking a timed test, and time is used up *)
      IF ((CURRSTRAT = TIMED) AND (ELASPEDTIME >= MAXTIME)) OR (TIMEOUT)  THEN
      BEGIN
        PAGE(OUTPUT);
        GOTOXY(1,8);
        WRITE('Your time has expired for this subtest.');
        GOTOXY(0,10);
        FLOWCODE := 5;
        STALL;
        TESTS.MODSTTIME := MAXTIME;
        EXIT(ADMINISTERQUESTION);
      END;


      CASE ABS(ACODE) OF
        (* wants to leave test *)
        5,7 : EXIT(ADMINISTERQUESTION);

        (* wants to leave session *)
        6   : DONE := TRUE;
      END;

    UNTIL (DONE) AND (ACODE = 1);
    (* if examinee wants to leave session, must first answer *)
    (* the question.  If wants to leave subtest, doesnt have *)
    (* to answer the question.                               *)
  END
  ELSE
  BEGIN
    PAGE(OUTPUT);
    SQUAWK;
    WRITELN('Question load error !!! No such question in directory !');
    WRITELN;
    WRITELN('Tried to load question # ',QSTCODE);
    WRITELN('Directory slot ', QSLOT);
    WRITELN;
    STALL;
  END;
END;  (* administer question *)
```

```
(****************************************************************************)
(*                                                                          *)
(*      Textfile : ADMIN.DIR/A.UABIL.TEXT       Volume : TFILES             *)
(*      Codefile : ADMIN.CODE ('Include' file)  Volume : CATDATA            *)
(*                                                                          *)
(****************************************************************************)
(*               DEC.  1, 1982                  NPROC                       *)
(****************************************************************************)


(* updates examinee ability level and variance of estimate *)
PROCEDURE UPDATEABILITY(PREFETCH : BOOLEAN);


    (* computes the examinees ability based on previous ability, *)
    (* variance, question abc parameters and how he answered     *)
    (* the question.                                              *)
    PROCEDURE BAYSIAN(VAR ABILITY,VARIANCE : REAL; A,B,C : REAL;
                      ANSWEROK : BOOLEAN);
    VAR SS,S,X,R,P,D : REAL;

        (*************************************************************)
        (* computes y = p(x) = probability that the random variable u, *)
        (* distributed norally (0,1), is less than or equal to x.  f(x)*)
        (* the ordinate of the normal density at x, is also computed   *)
        (*                                                             *)
        (* discription of parameters : xcoord -- input scalar for which*)
        (*                             p(x) is computed.               *)
        (*                             probability---output probability*)
        (*                             density--output density         *)
        (* minimum error is .0000007.                                  *)
        (* method : based on approximations in c. hastings,            *)
        (*          approximations for digital computers, princeton    *)
        (*          univ press, n.j., 1955, see equation 26.2.17,      *)
        (*          handbook of mathematical functions, abramowitz and *)
        (*          stegun, dover publications, inc., new york.        *)
        (*************************************************************)

        PROCEDURE NORMALDENSITY(XCOORD : REAL; VAR PROBABILITY,DENSITY : REAL);
        VAR AX,
            TEMP,
            AVGSQ : REAL;
        BEGIN
          AX := ABS(XCOORD);
          TEMP := 1.0 / (1.0 + 0.2316419 * AX);
          AVGSQ := -XCOORD * XCOORD / 2.0;
          DENSITY := 0.3989423 * EXP(AVGSQ);
          PROBABILITY := 1.0 - DENSITY * TEMP *
                         ((((1.330274 * TEMP - 1.821256) * TEMP + 1.781478)
                         * TEMP - 0.3565638) * TEMP + 0.3193815);
          IF XCOORD < 0 THEN
             PROBABILITY := 1.0 - PROBABILITY;
        END;  (* normal density *)


    BEGIN  (* bayeian *)
      SS := VARIANCE + 1.0 / SQR(A);
      S := SQRT(SS);
      X := (ABILITY - B) / S;
      NORMALDENSITY(X,P,D);
      IF ANSWEROK THEN
        R := D / (P + C / (1.0 - C))
      ELSE
        R := -D / (1.0 - P);
      ABILITY := ABILITY + R * VARIANCE / S;
      VARIANCE := VARIANCE - SQR(VARIANCE) * R * (R + X) / SS;
    END;   (* baysian *)

BEGIN  (* update ability *)
```

```
  {
  if trace then
  begin
    writeln;
    writeln('Old ability = ',currability);
    writeln('Old variance = ',currvariance);
  end;
  }

  CASE CURRSTRAT OF
    NONE,
    TIMED       : ;
    B102222,
    B54321,
    B108642     : BEGIN
                    IF PREFETCH THEN
                    BEGIN
                      BAYSIAN(RITEABILITY,
                              RITEVARIANCE,
                              ITEMINFO.A,ITEMINFO.B,ITEMINFO.C,
                              TRUE);
                      BAYSIAN(WRONGABILITY,
                              WRONGVARIANCE,
                              ITEMINFO.A,ITEMINFO.B,ITEMINFO.C,
                              FALSE);
                    END
                    ELSE
                    BEGIN
                      BAYSIAN(CURRABILITY,
                              CURRVARIANCE,
                              ITEMINFO.A,ITEMINFO.B,ITEMINFO.C,
                              TESTS.ITEMINFO[QNUM].CORRECT);
                      TESTS.ITEMINFO[QNUM].THETA := CURRABILITY;
                      TESTS.ITEMINFO[QNUM].ERROR := CURRVARIANCE;
                    END;
                  END;
  END;
  {
  if trace than
  begin
    writeln;
    writeln('A parameter = ',iteminfo.a);
    writeln('B parameter = ',iteminfo.b);
    writeln('C parameter = ',iteminfo.c);
    writeln;
    if tests.iteminfo[qnum].correct then
      writeln('Answered correctly : yes')
    else
      writeln('Answered correctly : no');
    writeln;
    writeln('New ability = ',tests.iteminfo[qnum].theta);
    writeln('New variance = ',tests.iteminfo[qnum].error);
    writeln;
    stall;
  end;
  }
END;    (* update ability *)
```

101

```
Apr  4 10:40 1983  ADMIN.DIR/A.FBACK.TEXT ( Give feedback to the examinee) Page 1

(*****************************************************************************)
(*                                                                           *)
(*      Textfile : ADMIN.DIR/A.FBACK.TEXT        Volume : TFILES             *)
(*      Codefile : ADMIN.CODE ('Include' file)   Volume : CATDATA            *)
(*                                                                           *)
(*****************************************************************************)
(* File last modified : Jan 28, 1983             NPROC                       *)
(*****************************************************************************)

SEGMENT PROCEDURE FEEDBACK(FBTYPE : TYPEFEEDBACK;
                           FBCODE, OUTPUTCODE : INTEGER);


     (* item feedback *)
     PROCEDURE ITEMFEEDBACK;
     VAR DUMMYCHAR : CHAR;
     BEGIN
       CASE FBCODE OF

         1 : BEGIN
               BLANKLINES(21,3,21);
               WRITE('The correct answer is ',ITEMINFO.ANSWER);
               GOTOXY(0,23);
               STALL;
             END;

         2 : ; (* remedial feedback *)

       END;  (* case *)
     END; (* item feedback *)


     (* subtest feedback *)
     PROCEDURE SUBTFEEDBACK(SUBF,SUBOUT : INTEGER);
     VAR TORDER : STRING;


          (* send detailed or simple feedback to printer or screen *)
          PROCEDURE OUTPUTRESULTS(PRINTER : BOOLEAN;
                                  E : EXAMEINFO;
                                  T : SUBTEST;
                                  STNUM : INTEGER);
          VAR I : INTEGER;


                   (* graphical detailed feedback *)
                   PROCEDURE GRAPHDISPLAY;
                   VAR X : INTEGER;
                       ANSWER : CHAR;
                       RANK : REAL;


                       (* calculate percentile rank based on ability *)
                       PROCEDURE PERCENTILE(XCOORD : REAL;
                                            VAR PROBABILITY : REAL);
                       VAR AX,
                           TEMP,
                           DENSITY,
                           AVGSQ : REAL;
                       BEGIN
                         AX := ABS(XCOORD);
                         TEMP := 1.0 / (1.0 + 0.2316419 * AX);
                         AVGSQ := -XCOORD * XCOORD / 2.0;
                         DENSITY := 0.3989423 * EXP(AVGSQ);
                         PROBABILITY := 1.0 - DENSITY * TEMP *
                                      ((((1.330274 * TEMP - 1.821256) * TEMP + 1.781478)
                                      * TEMP - 0.3565638) * TEMP + 0.3193815);
                         IF XCOORD < 0 THEN
                            PROBABILITY := 1.0 - PROBABILITY;
                         PROBABILITY := PROBABILITY * 100.0;
                       END;  (* percentile     *)
```

```
(* plots the graph *)
PROCEDURE EPLOT;
VAR POSITION : PACKED  ARRAY[-30..30] OF CHAR;
    EST      : INTEGER;
    Z        : INTEGER;
    VARIANCE : INTEGER;
    NUM      : INTEGER;
    THETA10  : REAL;
    ERROR10  : REAL;

 BEGIN  (* eplot *)
   NUM := 0;
   FOR X := 0 TO QUESTIONS DO
   BEGIN
     IF T.ITEMINFO[X].ITEMNUM >= 0 THEN
     WITH T.ITEMINFO[X] DO
     BEGIN
       NUM := NUM + 1;
       FILLCHAR(POSITION,61,' ');
       THETA10 := THETA * 10;
       EST := ROUND(THETA10);
       IF CORRECT = TRUE THEN
         POSITION[EST] := 'X'
       ELSE
         POSITION[EST] := 'O';
       ERROR10 := ERROR * 10;
       VARIANCE := ROUND(ERROR10);
       FOR Z := 1 TO VARIANCE DO
       BEGIN
         POSITION[EST+Z] := '.';
         POSITION[EST-Z] := '.';
       END;
       WRITE(DEST,POSITION);
       WRITE(DEST,THETA:4:2);
       WRITE(DEST,' ',ERROR:4:2);
       PERCENTILE(THETA,RANK);
       WRITELN(DEST,' ',RANK:3:1);
     END; (* with *)
   END; (* for [illegible]
   IF PRINTER THEN
   BEGIN
     WRITELN(DEST);
     WRITELN(DEST,' ':20,NUM,
            ' Items were administered');
     WRITELN(DEST);
   END;
 END;    (* eplot *)


BEGIN  (* graphdisplay *)
  IF PRINTER THEN
  BEGIN
    WRITE(DEST,' ':24,'Report on ');
    WRITE(DEST,TORDER);
    WRITELN(DEST,' ','Test');
    WRITE(DEST,'ID Number: ');
    WRITE(DEST,E.ID);
    WRITELN(DEST);
    WRITELN(DEST,' ':24,'X = Correct', ' ':12,
           'O = Incorrect');

    WRITELN(DEST,' ':7,
    'Error Band Plotted is + and - Standard Deviation');
    WRITELN(DEST);
  END;

  IF PRINTER THEN
    WRITELN(DEST,
    ' ':25, 'Ability Level')
  ELSE
```

```
      WRITELN(DEST,' Test : ',TORDER,
        '      Ability Level  (X=right, 0=wrong)');

    WRITELN(DEST,

 '   -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5',
               '            Ability  ');
    WRITELN(DEST,
  'I----I----I----I----I----I----I----I----I----I----I----',
             'I----I  Est  Var  Rank %');

    EPLOT;
  END;      (* graphdisplay *)


  (* simple summary *)
  PROCEDURE SIMPLESUMMARY;
  BEGIN
     WRITELN(DEST);
     WRITE(DEST,' ':24,'Report on ');
     WRITE(DEST,TORDER);
     WRITELN(DEST,' ','Test');
     WRITELN(DEST);
     WRITELN(DEST,' Examinee : ',E.ID);
     (WRITE(DEST,' # of screens in test : ',
            EXAMINEE.TESTLENGTH[STNUM]);)
     WRITE(DEST,'    # of items answered : ',T.NUMITEMS);
     WRITELN(DEST,'    # correct : ',T.NUMCORR);
     WRITELN(DEST);
  END;  (* simple summary *)


  (* table the results *)
  PROCEDURE TABLEDISPLAY;
  BEGIN
    IF PRINTER THEN
      SIMPLESUMMARY;
    IF NOT PRINTER THEN
    BEGIN
      GOTOXY(28,1);
      WRITE('     Report on ');
      GOTOXY(28,2);
      WRITE(TORDER);
      GOTOXY(26,5);
      (WRITE('# of screens in test : ',
            EXAMINEE.TESTLENGTH[STNUM]);)
      GOTOXY(26,6);
      WRITE('# of items answered : ',T.NUMITEMS);
      GOTOXY(26,7);
      WRITE('# correct : ',T.NUMCORR);
      GOTOXY(26,10);
    END;
  END;    (* table display *)



BEGIN  (* outputresults *)
  IF T.NUMITEMS > 0 THEN
  BEGIN

    IF PRINTER THEN
      REWRITE(DEST,UNITNUMPRINTER)
    ELSE
      REWRITE(DEST,'CONSOLE : ');

    IF NOT PRINTER THEN
      PAGE(OUTPUT);

    RESET(FILEDIRECTORY,INDEXNAME);
    SEEK(FILEDIRECTORY,SPARAMS.SUBORDER[STNUM]);
```

```
              GET(FILEDIRECTORY);
              TORDER := FILEDIRECTORY^.TESTNAME;
              CLOSE(FILEDIRECTORY,LOCK);

              CASE CURRSTRAT OF
                NONE : SIMPLESUMMARY;
                B102222,
                B54321,
                B108642 : GRAPHDISPLAY;
                TIMED : TABLEDISPLAY;
              END;

              IF NOT PRINTER THEN
                STALL;
              CLOSE(DEST,NORMAL);
            END;
          END;   (* outputresults *)


     BEGIN  (* subtest feedback *)
       CASE SUBOUT OF
         1    : OUTPUTRESULTS(FALSE,EXAMINEE,TESTS,TINDEX);
         2,3  : OUTPUTRESULTS(TRUE,EXAMINEE,TESTS,TINDEX);
         4,5  : BEGIN
                  OUTPUTRESULTS(FALSE,EXAMINEE,TESTS,TINDEX);
                  OUTPUTRESULTS(TRUE,EXAMINEE,TESTS,TINDEX);
                END;
       END;
     END;  (* subtestfeedback *)



     (* session feedback *)
     PROCEDURE SESSIONFEEDBACK;
     VAR RSLOT : INTEGER;
     BEGIN

       (* get location of first subtest *)
       RSLOT := ERECNUM * GMAXSUBTEST;

       (* display results of each subtest *)
       FOR TINDEX := 1 TO GMAXSUBTEST DO
       BEGIN

         (* get strategy to display appropriate feedback *)
         CURRSTRAT := EXAMINEE.STRATEGY[TINDEX];

         (* load results into record *)
         LOADRESULTS(RSLOT);

         (* if any questions answered *)
         IF TESTS.NUMITEMS > 0 THEN
         BEGIN
           CASE OUTPUTCODE OF
             1 : SUBTFEEDBACK(1,1);
             2 : SUBTFEEDBACK(1,2);
             3 : SUBTFEEDBACK(1,4);
           END;
         END;
         RSLOT := RSLOT + 1;
       END;  (* for *)
     END;  (* session feedback *)



BEGIN (* feedback *)


  IF OUTPUTCODE  <> 0 THEN
  BEGIN
  (* cannot give session feedback to demos *)
  IF (DEMOFLAG) AND (FBTYPE = FBSESSION)  THEN
   EXIT(FEEDBACK);

  IF (FBTYPE = FBSUBTEST) OR (FBTYPE = FBSESSION) THEN
  BEGIN
   TEXT80MODE;
   INVERSE;
  END;
  CASE FBTYPE OF
   FBITEM    : ITEMFEEDBACK;
   FBSUBTEST : SUBTFEEDBACK(FBCODE,OUTPUTCODE);
   FBSESSION : SESSIONFBACK;
  END;
  IF (FBTYPE = FBSUBTEST) OR (FBTYPE = FBSESSION) THEN
  BEGIN
   TEXT40MODE;
   INVERSE;
  END;
 END;

END;  (* feedback *)
```

```
(**********************************************************************)
(*                                                                    *)
(*    Textfile : ADMIN.DIR/A.PASVAB.TEXT              Volume : TFILES *)
(*    Codefile : A.PASVAB.CODE                        Volume : CATDATA*)
(*                                                                    *)
(**********************************************************************)
(*              June 10, 1983                         NPRDC           *)
(**********************************************************************)
(*  This function receives a theta (CURRABILITY) level for a certain  *)
(*  subtest, and returns a value representing the predicted ASVAB   *)
(*  score for a pencil/paper test. The score is obtained through a  *)
(*  'table look-up' from a file of records corresponding to a particular *)
(*  subtest.  The lookup is done by using the theta value as a key  *)
(*  to the appropriate record in the file. For example, if theta=2.70 or -2.70*)
(*  then slot #7 in record #2 will contain the score to be accessed.  *)
(*  Each of the score slots correspond to theta values 0.1 apart, e.g. *)
(*  thetas of 4.80,4.90, occupy record #4 slots #8,9 respectively. For *)
(*  thetas to the 0.01 place, two lookups will take place, one for the *)
(*  slot below and for the slot above- for theta of 2.392 record #2    *)
(*  3 and 4 will be accessed. This will be followed by a simple linear *)
(*  interpolation of the two values resulting in an approximate score *)
(*  accurate to +-0.01.                                                *)


FUNCTION PREASVABCALC (THETA:REAL):REAL;


CONST
  MAXREC=5;
  MAXSLOT=9;

TYPE
    THETAREC=RECORD
         P:PACKED ARRAY[0..MAXSLOT]OF REAL;  (* P- positive thetas *)
         N:PACKED ARRAY[0..MAXSLOT]OF REAL;  (* N- negative thetas *)
       END;
VAR
   SCOREC:THETAREC;
   SCORE,Z:REAL;
   C,I,J,Y:INTEGER;
   SCOREFILE:FILE OF THETAREC;
   CONT,POS:BOOLEAN;
   HIGH,LOW,THETA1:REAL;


       FUNCTION  INTERP(LOW,HIGH:REAL):REAL;

       BEGIN

       SCORE:= (THETA - THETA1)/0.1;        (* INTERPOLATE *)
       INTERP:= (SCORE * (HIGH-LOW)) + LOW;
       END; (* interp *)


       PROCEDURE SEEKREC(Y:INTEGER);
       VAR X:INTEGER;

       BEGIN
          FOR X:= 0 TO Y  DO      (* SIMULATE SEEK FUNCTION *)
          GET(SCOREFILE);
          SCOREC :=SCOREFILE^;
          RESET(SCOREFILE);
       END;
            (* seekrec *)


         PROCEDURE GETSCORE;

         BEGIN  (* getscore *)

           IF POS THEN
          BEGIN
          SCORE:=SCOREC.P[C];
          IF Y=MAXREC THEN
          POS:=FALSE;     (* IF POS ASSERTED, CHANGE BACK *)
          END
          ELSE
          SCORE:=SCOREC.N[C];
        END;  (* getscore *)


      PROCEDURE GETADDRESS(VAR THETA:REAL);

      BEGIN  (* getaddress *)

        IF THETA < 0 THEN POS:=FALSE;
        THETA:= ABS(THETA);
        Y:=TRUNC(THETA);
        Z:=THETA - Y;
        Z:=(Z * 10)+ 0.000444;
        C:=TRUNC(Z);
        IF (NOT POS) AND (Y >= 6)       (* THETA IS -6; GOTO RECORD #5   *)
        THEN
        BEGIN
          POS:=TRUE;                    (* CHANGE POS TO GET LAST REC *)
          Y := MAXREC;
          IF THETA1 >= 6.9 THEN
          BEGIN
          C := MAXSLOT;
          CONT:=FALSE;
          END;
        END
        ELSE IF POS AND (THETA1 >= 4.9)
        THEN
        BEGIN
          Y:=MAXREC-1;
          C:=MAXSLOT;
          CONT:=FALSE;
         END;
      END; (* getaddress *)


     PROCEDURE LOOKUP(VAR THETA:REAL);

        BEGIN      (* lookup *)

        GETADDRESS(THETA);
        SEEKREC(Y);
        GETSCORE;
        IF CONT THEN
          BEGIN
           LOW:=SCORE;
           HIGH:=THETA1 + 0.1;
           GETADDRESS(HIGH);
           SEEKREC(Y);
           GETSCORE;
           HIGH:=SCORE;
           SCORE:=INTERP(LOW,HIGH)
          END;
          CLOSE(SCOREFILE);
END;     (* lookup *)


BEGIN     (* preasvabcalc *)

    CASE CURRTEST OF
      0:  RESET(SCOREFILE,'/CATDATA/WKTABLE.DATA');
      1:  RESET(SCOREFILE,'/CATDATA/GSTABLE.DATA');
      2:  RESET(SCOREFILE,'/CATDATA/ARTABLE.DATA');
      3:  RESET(SCOREFILE,'/CATDATA/MKTABLE.DATA');
      4:  RESET(SCOREFILE,'/CATDATA/PCTABLE.DATA')
      OTHERWISE EXIT(PREASVABCALC)
    END;  (*case*)
    CONT:=TRUE;
    POS:=TRUE;
    THETA1:= ABS(TRUNC(THETA * 10)/10); (* preserve tenths place;truncate rest*)
    IF ABS(THETA)=THETA1
    THEN CONT:=FALSE;
    LOOKUP(THETA);

    PREASVABCALC:= SCORE;
END;    (* preasvabcalc *)
```

PMGR.DIR:
Subdirectory - Parameter Management Textfiles

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : PMGR.DIR/P.MGR.TEXT          Volume : TFILES               *)
(*      Codefile : P.MGR.CODE                   Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(* File last modified : Jan 31, 1983            NPROC                         *)
(******************************************************************************)
(*                                                                            *)
(* This program sets up one file for the CAT program. The file is the testing *)
(* strategies & parameters given to examinees.                                *)
(*                                                                            *)
(******************************************************************************)

(*$S+*)

PROGRAM SYSTEMSETUP;
USES CHAINSTUFF;

CONST (* ascii values *)
      BELL = 7;
      LARROW = 8;
      RARROW = 21;
      RET = 13;
      ESC = 27;
      SPACE = 32;
      ASCIIOFFSET = 48;
      NIL = -1;


      (* default file for any output text files *)
      DEFAULTFILE = 'CAT-SETUP.TEXT';

      (* test configuration file *)
      SETUPDATA = 'CATDATA:PARAMETERS.DATA'; (* test default parameters *)

      (* subtest directory *)
      INDEXNAME = 'CATDATA:TESTINDEX.DATA';

      MAXITEMPOOL = 300;

      (* maximum allowable subtests given *)
      GMAXSUBTEST = 20;

      (* maximum spaces in directory *)
      MAXSUBTESTS = 20;

      (* printer unit number *)
      UNITNUMPRINTER = 'printer:';

      VERSION = '[1.03]';

TYPE SETOFCHAR = SET OF CHAR;

     (* test directory *)
     DIRDATA = PACKED RECORD
                        UNUSED      : BOOLEAN;
                        TESTNAME    : STRING;
                        ITEMCODE    : PACKED ARRAY
                                       [0..MAXITEMPOOL]
                                      OF INTEGER;
                      END;


     (* set-up parameters *)
     SETUPINFO = PACKED RECORD

                   (* flags # tests in sequence, > 20 = no subtest *)
                   SUBORDER,

                   (* strategy setup *)
                   SUBSTRAT        : PACKED ARRAY[1..GMAXSUBTEST]
                                    OF 0..128;
                   (* question feedback code *)
                   ITEMFB,

                   (* question feedback output code *)
                   ITEMOUTPUT,

                   (* subtest feedback code *)
                   SUBTESTFB,

                   (* subtest feedback destination code *)
                   SUBTESTOUTPUT,

                   (* session feedback code *)
                   SESSIONFB,

                   (* session feedback destination; screen/printer *)
                   SFSSIONOUTPUT  : 0..128;

                   (* subtest stop flag *)
                   SUBSTOP           : PACKED ARRAY[1..GMAXSUBTEST] OF BOOLEAN;

                   (* subtest length *)
                   SUBLENGTH         : PACKED ARRAY[1..GMAXSUBTEST]
                                       OF 0..128;

                   (* initial variance *)
                   CKERROR           : ARRAY[1..GMAXSUBTEST] OF REAL;

                 END;


VAR output,
    COMMAND : CHAR;

    DIGITS,
    LETTERS,
    CHARACTERS : SETOFCHAR;

    TEMPNAMEORDER,
    NAMEORDER       : ARRAY[1..GMAXSUBTEST] OF STRING;


    (* string buffer *)
    LINEBUF : PACKED ARRAY[0..79] OF CHAR;


    (* test directory *)
    DIRECTORY : DIRDATA;
    FILEDIRECTORY : FILE OF DIRDATA;


    (*** set-up variables ***)
    SPARAMS : SETUPINFO;
    FILESPARAMS : FILE OF SETUPINFO;


    (* output text file variable *)
    DEST : TEXT;

    (* directory record information *)
    DIRINFO : ARRAY[0..MAXSUBTESTS] OF RECORD

                                          (* record occupied *)
                                          NOTUSED : BOOLEAN;

                                          (* subtest name *)
                                          TNAME : STRING;

                                          (* # items in subtest *)
                                          ITEMCOUNT : INTEGER;
                                        END;
```

```
(* ................................................................... *)

(* writes the directory information to record    *)
(* puts necessary file info in main memory       *)
PROCEDURE GETDIRINFO;
VAR I,K,ICOUNT : INTEGER;
BEGIN

  (* initialize the directory information *)
  FOR I := 0 TO MAXSUBTESTS DO
    DIRINFO[I].NOTUSED := TRUE;

  (* get the directory information *)
  I := 0;
  RESET(FILEDIRECTORY,INDEXNAME);
  REPEAT
    SEEK(FILEDIRECTORY,I);
    GET(FILEDIRECTORY);
    IF NOT (FILEDIRECTORY^.UNUSED) THEN
    BEGIN
      DIRECTORY := FILEDIRECTORY^;
      DIRINFO[I].NOTUSED := FALSE;
      DIRINFO[I].TNAME := DIRECTORY.TESTNAME;
      DIRINFO[I].ITEMCOUNT := 0;
    END;
    I := I + 1;
  UNTIL I > MAXSUBTESTS;
  CLOSE(FILEDIRECTORY,NORMAL);
END;  (* getdirinfo *)

(* ------------------------------------------------------------------- *)

(* clear screen and put cursor at 0,0 *)
PROCEDURE PAGE(DUMMY : CHAR);
BEGIN
  WRITE(CHR(26));
  GOTOXY(0,0);
END;  (* page *)

(* ------------------------------------------------------------------- *)

(*$I /TFILES/PMGR.DIR/P.UTL.TEXT *)      (* utilities *)
(*$I /TFILES/PMGR.DIR/P.VIEW.TEXT *)     (* look at current configuration *)
(*$I /TFILES/PMGR.DIR/P.SP.TEXT *)       (* set up test configuration *)
(*$I /TFILES/PMGR.DIR/P.FBACK.TEXT *)    (* set up feedback/output *)

(* ------------------------------------------------------------------- *)

(* system set-up menu *)
PROCEDURE SETUPMENU;
VAR X,Y : INTEGER;
BEGIN
  PAGE(OUTPUT);
  GOTOXY(15,0);
  WRITE('CONFIGURE TEST PARAMETERS MENU ',VERSION);
  GOTOXY(0,4);
  WRITE('Select one of the following procedures by entering its number.');
  X := 16;
  Y := 8;
  GOTOXY(X,Y);
  WRITE('1.  QUIT');
  GOTOXY(X,Y+1);
  WRITE('2.  LOOK AT CURRENT PARAMETERS');
  GOTOXY(X,Y+2);
  WRITE('3.  MODIFY SYSTEM PARAMETERS');
  GOTOXY(X,Y+3);
  WRITE('4.  CONFIGURE FEEDBACK PARAMETERS');
  GOTOXY(X,Y+7);
  WRITE('Enter Choice # : ');
END;  (* setup menu *)

(* ------------------------------------------------------------------- *)
```

```
    (* load the set-up parameters *)
    PROCEDURE LOADPARAMS;
    VAR I,J : INTEGER;
    BEGIN
      RESET(FILESPARAMS,SETUPDATA);
      SEEK(FILESPARAMS,0);
      GET(FILESPARAMS);
      SPARAMS := FILESPARAMS^;
      CLOSE(FILESPARAMS,LOCK);
      J := 1;
      RESET(FILEDIRECTORY,INDEXNAME);
      REPEAT
        I := SPARAMS.SUBORDER[J];
        IF (I <= MAXSUBTESTS) THEN
        BEGIN
          SEEK(FILEDIRECTORY,I);
          GET(FILEDIRECTORY);
          NAMEORDER[J] := FILEDIRECTORY^.TESTNAME;
        END
        ELSE
          NAMEORDER[J] := '';
        J := J + 1;
      UNTIL J > GMAXSUBTEST;
      CLOSE(FILEDIRECTORY,LOCK);
    END;   (* loadparams *)

    (* ------------------------------------------------------------------------ *)

    (* save the set-up parameters *)
    PROCEDURE SAVEPARAMS;
    BEGIN
      RESET(FILESPARAMS,SETUPDATA);
      SEEK(FILESPARAMS,0);
      FILESPARAMS^ := SPARAMS;
      PUT(FILESPARAMS);
      CLOSE(FILESPARAMS,LOCK);
    END;   (* saveparams *)

(* ............................................................................ *)

(**** main program ***)
BEGIN
  GETDIRINFO;
  DIGITS := ['0'..'9'];
  LETTERS := ['A'..'Z','a'..'z'];
  CHARACTERS := [CHR(32)..CHR(126)];
  FILLCHAR(LINEBUF[0],80,' ');
  LOADPARAMS;
  REPEAT
    SETUPMENU;
    COMMAND := GETCHAR(['1'..'4'],TRUE,FALSE,TRUE);
    CASE COMMAND OF
      '1' : ;
      '2' : VIEWPARAMS;
      '3' : SETPARAMETERS;
      '4' : SETFBACK;
    END;
  UNTIL COMMAND = '1';
  SAVEPARAMS;
  SETCHAIN('CATDATA:CATPROJECT');
END.  (* P.MGR *)
```

```
(********************************************************************************)
(*                                                                             *)
(*      Textfile : PMGR.DIR/P.UTL.TEXT              Volume : TFILES            *)
(*      Codefile : P.MGR.CODE                       Volume : CATDATA           *)
(*                                                                             *)
(********************************************************************************)
(*                 DEC.  1, 1982                    NPROC                      *)
(********************************************************************************)

(* This file contains the utilities used in program P.MGR. *)

(* form feeds the printer *)
PROCEDURE TOPOFFORM;
BEGIN
  REWRITE(DEST,UNITNUMPRINTER);
  WRITE(DEST,CHR(12));
  CLOSE(DEST,LOCK);
END;   (* top of form *)

(* -----------------------------------------------------------------------------*)

(**** rings the bell ****)
PROCEDURE SQUAWK;
BEGIN
  WRITE(CHR(BELL));
END;  (* squawk *)

(* -----------------------------------------------------------------------------*)

(**** blank out lines  ****)
PROCEDURE BLANKLINES(START,COUNT,ENDCURSOR : INTEGER);
VAR I : INTEGER;
BEGIN
  GOTOXY(0,START);
  FOR I := 1 TO (COUNT-1) DO
    WRITELN(' ' : 39);
  WRITE(' ':39);
  GOTOXY(0,ENDCURSOR);
END;   (* blanklines *)

(* -----------------------------------------------------------------------------*)


(* read an acceptable character from the keyboard *)
FUNCTION GETCHAR(OKSET : SETOFCHAR;
                 FLUSHQUEUE,ECHO,BEEP : BOOLEAN) : CHAR;
VAR MASK : PACKED ARRAY[0..0] OF CHAR;
BEGIN
  IF FLUSHQUEUE THEN UNITCLEAR(2); (* flush buffer *)
  REPEAT
    UNITREAD(2,MASK,1);
    IF BEEP AND NOT (MASK[0] IN OKSET) THEN SQUAWK;

  UNTIL MASK[0] IN OKSET;
  IF ECHO AND (MASK[0] IN [CHR(32)..CHR(126)]) THEN
    WRITE(MASK[0]);
  GETCHAR := MASK[0];
END;  (* getchar *)

(* -----------------------------------------------------------------------------*)

(**** display a message/wait for a keystroke ****)
PROCEDURE STALL;
VAR STALLCHAR : CHAR;
BEGIN
  WRITE('Press <RET> to continue ');
  STALLCHAR :=
    GETCHAR([CHR(RET),CHR(ESC)],TRUE,FALSE,TRUE);
  IF STALLCHAR = CHR(ESC) THEN EXIT(PROGRAM);
END;  (* stall *)

(* -----------------------------------------------------------------------------*)
(* read in a string and save in a temporary buffer  *)
PROCEDURE FILLBUF(CHARCNT : INTEGER;
                  OKSET : SETOFCHAR; ERASE : BOOLEAN);
VAR I : INTEGER;
    IDCHAR : CHAR;
BEGIN
  I := 0;
  REPEAT
    IF I > (CHARCNT-1) THEN
      IDCHAR :=
        GETCHAR([CHR(LARROW),CHR(RET)],TRUE,TRUE,TRUE)
    ELSE
    BEGIN (1)
      IDCHAR :=
        GETCHAR(OKSET + [CHR(RET),
                CHR(LARROW),CHR(RARROW)],
                TRUE,TRUE,TRUE);
      IF IDCHAR IN OKSET THEN
      BEGIN (2)
        LINEBUF[I] := IDCHAR;
        I := I + 1;
      END;  (2)
    END;  (1)
    IF IDCHAR = CHR(LARROW) THEN
    BEGIN (3)
      IF I = 0 THEN
        SQUAWK
      ELSE
      BEGIN (4)
        WRITE(CHR(LARROW));
        I := I - 1;
        IF ERASE THEN
        BEGIN (5)
          WRITE(' ');
          WRITE(CHR(LARROW));
          LINEBUF[I] := ' ';
        END;  (5)
      END;  (4)
    END   (3)
    ELSE
      IF IDCHAR = CHR(RARROW) THEN
      BEGIN (6)
        WRITE(LINEBUF[I]);
        I := I + 1;
      END;  (6)
  UNTIL IDCHAR = CHR(RET);
END;  (* fillbuf *)

(* -----------------------------------------------------------------------*)

(**** open a new text file ****)
PROCEDURE  GETNEWFILE;
VAR FILENAME : STRING;
    ERRNUM : INTEGER;

    (* ............................................................... *)

    (**** get a legal filename ****)
    FUNCTION NAMEOK : BOOLEAN;
    VAR I : INTEGER;
    BEGIN
      IF FILENAME = '' THEN
      BEGIN (1)
        FILENAME := DEFAULTFILE;
        GOTOXY(44,0);
        WRITE(FILENAME);
      END   (1)
      ELSE
        IF FILENAME[1] = CHR(esc) THEN EXIT(PROGRAM);
      IF (POS('.TEXT',FILENAME) <> (LENGTH(FILENAME) - 4))
         OR (LENGTH(FILENAME) < 6 ) THEN
           FILENAME := CONCAT(FILENAME,'.TEXT');
    (*$I-*)
```

```
      RESET(DEST,FILENAME);
    (*$I+*)
      IF IORESULT = 0 THEN
      BEGIN {2}
        WRITELN;
        WRITELN;
        WRITE('Destroy old ',FILENAME,'?  Press ''N'' or ''Y'' ');
        IF GETCHAR(['y','n','Y','N'],TRUE,TRUE,TRUE) IN ['Y','y'] THEN
        BEGIN {3}
          CLOSE(DEST,PURGE);
          REWRITE(DEST,FILENAME);
          NAMEOK := TRUE;
        END   {3}
        ELSE
          NAMEOK := FALSE;
      END   {2}
      ELSE
      BEGIN {4}
      (*$I-*)
        REWRITE(DEST,FILENAME);
      (*$I+*)
        ERRNUM := IORESULT;
        IF IORESULT <> 0 THEN
        BEGIN {5}
          WRITELN;
          WRITELN;
          WRITELN('Cannot open ',FILENAME,' Io error #',ERRNUM);
          NAMEOK := FALSE;
        END   {5}
        ELSE
          NAMEOK := TRUE;
      END;  {4}
    END;  (* nameok *)

(* ....................................................................... *)

BEGIN  (* getnewfile  *)
  REPEAT
    PAGE(OUTPUT);
    WRITE('Enter output file name, then press <RET>  : ');
    READLN(FILENAME);
  UNTIL NAMEOK;
END;

(* -----------------------------------------------------------------------*)

(* send control characters to screen *)
PROCEDURE SCRCONTROL(I,J,K : INTEGER);  { PASCAL interface to Screen Control}
VAR N: INTEGER;                         { APPLE III Standard Device Drivers}
    G_ARRAY:PACKED ARRAY[0.. 3] OF 0..255;        {....... Pages 34 to 46.}
BEGIN
  G_ARRAY[0]:= I; G_ARRAY[1]:=J; G_ARRAY[2]:=K;
  UNITWRITE(1,G_ARRAY,3,,12);
END;   (* scrcontrol *)

(* -----------------------------------------------------------------------*)

(* switch to 40 column screen *)
PROCEDURE TEXT40MODE;
BEGIN
  SCRCONTROL(16,0,28); {Text mode 40BW, followed by clearscreen.}
END;  (* text40mode *)

(* -----------------------------------------------------------------------*)

(* switch to 80 column screen *)
PROCEDURE TEXT80MODE;
BEGIN
  SCRCONTROL(04,0,0); {Restore Viewport to its original condition.}
  SCRCONTROL(16,2,28);{ Text Mode 80, followed by clearscreen.}
END;   (* text80mode *)

(* -----------------------------------------------------------------------*)
(* turn on reverse video *)
PROCEDURE INVERSE;
BEGIN
  SCRCONTROL(18,0,0);
END;     (* inverse *)

(* ---------------------------------------------------------------------------*)

(* switch back to normal screen *)
PROCEDURE NORMAL;
BEGIN
  SCRCONTROL(17,0,0);
END;
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : PMGR.DIR/P.VIEW.TEXT         Volume : TFILES               *)
(*      Codefile : P.MGR.CODE ('include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(* File last modified : Jan 28, 1983            NPROC                         *)
(******************************************************************************)

(* view the current configuration *)
PROCEDURE VIEWPARAMS;
CONST EMPTY = '                                                   ';

VAR COUNT,
    I : INTEGER;
    SCREEN : BOOLEAN;

    (* ...................................................................... *)

    (* list out the tests *)
    PROCEDURE LISTTESTS;
    BEGIN
      WRITE(DEST,'                TEST            LENGTH    STRATEGY   ');
      WRITELN(DEST,'       CKERROR  STOPFLAG');
      FOR I := 1 TO GMAXSUBTEST DO
      BEGIN (1)
        IF SPARAMS.SUBORDER[I] <= MAXSUBTESTS THEN
        BEGIN (2)
          WRITE(DEST,NAMEORDER[I] : 30);
          WRITE(DEST,SPARAMS.SUBLENGTH[I] : 5);
          CASE SPARAMS.SUBSTRAT[I] OF
            0 : WRITE(DEST,'       None         ');
            1 : WRITE(DEST,'       B 102222     ');
            2 : WRITE(DEST,'       B 54321      ');
            3 : WRITE(DEST,'       B 108642     ');
            4 : WRITE(DEST,'       Timed        ');
          END; (* cases *)
          WRITE(DEST,SPARAMS.CKERROR[I] : 12 : 3);
          IF SPARAMS.SUBSTOP[I] THEN
            WRITE(DEST,'    Yes')
          ELSE
            WRITE(DEST,'    No');
          WRITELN(DEST);
        END;  (2)
      END;  (1)
    END;  (* listtests *)

    (* ---------------------------------------------------------------------- *)

    (* list the feedback configuration *)
    PROCEDURE LISTFBACK;
    BEGIN
      WRITELN(DEST);
      IF SPARAMS.ITEMOUTPUT = 0 THEN
        WRITE(DEST,'Item  : none    ')
      ELSE
      BEGIN (1)
        WRITE(DEST,'Item  : ');
        CASE SPARAMS.ITEMFB OF
          1 : WRITE(DEST,'right/wrong');
          2 : WRITE(DEST,'remedial');
        END; (* cases *)
      END;  (1)
      IF SPARAMS.SUBTESTOUTPUT = 0 THEN
        WRITE(DEST,'Subtest : none     ')
      ELSE
      BEGIN (2)
        WRITE(DEST,'Subtest : ');
        CASE SPARAMS.SUBTESTOUTPUT OF
          1 : WRITE(DEST,'screen      ');
          2 : WRITE(DEST,'printer     ');
          3 : WRITE(DEST,'screen & printer     ');
        END; (* cases *)
      END;   (2)
      IF SPARAMS.SESSIONOUTPUT = 0 THEN
        WRITE(DEST,'Session : none     ')
      ELSE
      BEGIN (3)
        WRITE(DEST,'Session : ');
        CASE SPARAMS.SESSIONOUTPUT OF
          1 : WRITE(DEST,'screen');
          2 : WRITE(DEST,'printer');
          3,4,5 : WRITE(DEST,'screen & printer');
        END; (* cases *)
      END;   (3)
      WRITELN(DEST);
    END;    (* listfback *)

(* ............................................................... *)

BEGIN  (* view params *)

  (*
    PAGE(OUTPUT);
    WRITE('List configuration to : ');
    GOTOXY(15,5);
    WRITE('1) Console');
    GOTOXY(15,7);
    WRITE('2) Printer');
    GOTOXY(24,0);
    IF GETCHAR(['1','2'],TRUE,TRUE,TRUE) = '2' THEN
      BEGIN (1)
        REWRITE(DEST,UNITNUMPRINTER);
        SCREEN := FALSE;
      END    (1)
      ELSE
      BEGIN (2)
        *)
      REWRITE(DEST,'CONSOLE:');
      SCREEN := TRUE;
    (* END;*)


   PAGE(OUTPUT);
   LISTTESTS;
   LISTFBACK;
   IF SCREEN THEN
     STALL;
   CLOSE(DEST,LOCK);
END;  (* view params *)
```

```
(*************************************************************************)
(*                                                                       *)
(*      Textfile : PMGR.DIR/P.SP.TEXT           Volume : TFILES          *)
(*      Codefile : P.MGR.CODE ('Include' file)  Volume : CATDATA         *)
(*                                                                       *)
(*************************************************************************)
(*  File last modified : Jan 20, 1983           NPROC                    *)
(*************************************************************************)

PROCEDURE SETPARAMETERS;

VAR COMMAND : CHAR;
    TEMP : SETUPINFO;

    (* ................................................................. *)

    PROCEDURE SELECT20;
    VAR I,
        J,
        K,
        LISTCNT,
        COUNT,
        CURSORLOC : INTEGER;
        COMMAND : CHAR;

        OK,
        FOUND : BOOLEAN;
        LISTINFO : ARRAY[0..MAXSUBTESTS] OF RECORD
                                              NAME : STRING;
                                              INLIST : BOOLEAN;
                                            END;

        FREE : ARRAY[1..GMAXSUBTEST] OF BOOLEAN;

        (* ............................................................. *)

        (* get list of available subtests & current ones *)
        PROCEDURE GETLIST;
        BEGIN
          I := 0;
          J := 0;
          REPEAT
            IF NOT (DIRINFO[J].NOTUSED) THEN
            BEGIN {1}
              LISTINFO[I].NAME := DIRINFO[J].TNAME;
              K := 1;
              FOUND := FALSE;
              REPEAT
                IF NAMEORDER[K] = LISTINFO[I].NAME THEN
                  FOUND := TRUE;
                K := K + 1;
              UNTIL (FOUND) OR (K > GMAXSUBTEST);
              IF FOUND THEN
                LISTINFO[I].INLIST := TRUE
              ELSE
                LISTINFO[I].INLIST := FALSE;
              I := I + 1;
            END;  {1}
            J := J + 1;
          UNTIL J > MAXSUBTESTS;
          LISTCNT := I-1;
        END;  (* getlist *)

        (* ------------------------------------------------------------- *)

        (* add a subtest to the list *)
        PROCEDURE ADDSUBTEST;
        VAR CONVERT : ARRAY[1..21] OF INTEGER;
            ADD : INTEGER;
        BEGIN
          PAGE(OUTPUT);
          IF COUNT < GMAXSUBTEST THEN
          BEGIN {1}
```

```
            K := 1;
            FOR I := 0 TO LISTCNT DO
              IF NOT LISTINFO[I].INLIST THEN
              BEGIN {2}
                GOTOXY(18,K+1);
                WRITE(K,'.  ',LISTINFO[I].NAME);
                CONVERT[K] := I;
                K := K + 1;
              END;  {2}
            IF K = 1 THEN
            BEGIN {3}
              WRITE(' ':18,'No tests available to add!');
              SQUAWK;
              WRITELN;
              WRITELN;
              STALL;
            END   {3}
            ELSE
            BEGIN {4}
              OK := FALSE;
              REPEAT
                FILLCHAR(LINEBUF[0],' ',2);
                GOTOXY(0,0);
                WRITE(
'Type in the number next to the test you wish to add and then press <RET>.');
                GOTOXY(18,23);
                WRITE('Enter Choice # : ');
                FILLBUF(2,['0'..'9'],TRUE);
                IF LINEBUF[1] IN ['0'..'9'] THEN
                  ADD := ((ORD(LINEBUF[0]) - 48) * 10)
                         + (ORD(LINEBUF[1]) - 48)
                ELSE
                  ADD := ORD(LINEBUF[0]) - 48;
                IF ADD = 0 THEN EXIT(ADDSUBTEST);
                IF (ADD < 0) OR (ADD > K) THEN
                  SQUAWK
                ELSE
                BEGIN {5}
                  OK := TRUE;
                  LISTINFO[CONVERT[ADD]].INLIST := TRUE;
                END;  {5}
              UNTIL OK;
            END;  {4}
          END   {1}
          ELSE
          BEGIN {6}
            SQUAWK;
            WRITE(' ':18,' Cannot add another test!');
            WRITELN;
            WRITELN;
            STALL;
          END;  {6}
        END;  (* addsubtest *)

        (* ------------------------------------------------------------------ *)

        (* normalize & fix the lists what we try to do is save the *)
        (* configuration of tests that were'nt removed.     *)
        PROCEDURE ADJUSTLIST;

        BEGIN

          (* save the 'free' indices in the array of tests order *)
          (* and remove from 'inlist' the things already in the  *)
          (* subtest order list, so we don't count them twice.   *)

          FOR I := 1 TO GMAXSUBTEST DO
          BEGIN {1}
            FREE[I] := TRUE;  (* initialize ith slot to unoccupied *)
            J := 0;
            FOUND := FALSE;

            (* see if ith subtest is in list *)
```

```
      REPEAT
        IF (LISTINFO[J].INLIST) AND
        (LISTINFO[J].NAME = NAMEORDER[I]) THEN
        BEGIN {2}
          FOUND := TRUE;
          FREE[I] := FALSE;   (* mark Ith location as occupied *)
          LISTINFO[J].INLIST := FALSE;   (* take out of list *)
        END;  {2}
        J := J + 1;
      UNTIL (FOUND) OR (J > LISTCNT);
      IF NOT FOUND THEN
        NAMEORDER[I] := '';
    END;  {1}

    (* put rest of inlist tests not already in subtest order array *)
    (* into the array                                               *)
    FOR I := 0 TO LISTCNT DO
    BEGIN {3}
      IF LISTINFO[I].INLIST THEN
      BEGIN {4}
        J := 1;
        FOUND := FALSE;

        (* find unoccupied slot to put in *)
        REPEAT
          IF FREE[J] THEN
          BEGIN {5}
            FOUND := TRUE;
            FREE[J] := FALSE;
          END   {5}
          ELSE
            J := J + 1;
        UNTIL (FOUND) OR (J > GMAXSUBTEST);

        (* no empty spots *)
        IF J > GMAXSUBTEST THEN
        BEGIN {6}
          PAGE(OUTPUT);
          WRITE('List formation error!');
          SQUAWK;
          WRITELN;
          WRITELN;
          STALL;
        END   {6}
        ELSE
          NAMEORDER[J] := LISTINFO[I].NAME;
      END;  {4}
    END;  {3}

    I := 1;
    FOR J := 1 TO GMAXSUBTEST DO
    BEGIN {7}
      IF NAMEORDER[J] <> '' THEN
      BEGIN {8}
        TEMP.SUBSTRAT[I] := SPARAMS.SUBSTRAT[J];
        TEMP.SUBSTOP[I] := SPARAMS.SUBSTOP[J];
        TEMP.SUBLENGTH[I] := SPARAMS.SUBLENGTH[J];
        TEMP.CKERROR[I] := SPARAMS.CKERROR[J];
        TEMPNAMEORDER[I] := NAMEORDER[J];
        TEMP.SUBORDER[I] := I;
        I := I + 1;
      END;  {8}
    END;  {7}
    WHILE I <= GMAXSUBTEST DO
    BEGIN {9}
      TEMP.SUBORDER[I] := 128;  (* mark as unused *)
      TEMPNAMEORDER[I] := '';
      I := I + 1;
    END;  {9}
    SPARAMS := TEMP;
    NAMEORDER := TEMPNAMEORDER;
  END;  (* adjust list *)
```

```
(* ................................................................... *)

BEGIN  (* select20 *)
  TEMP := SPARAMS;
  GETLIST;
  REPEAT
    CURSORLOC := 0;
    COUNT := 0;
    PAGE(OUTPUT);
    WRITELN('     Tests Currently in List');
    FOR I := 1 TO 40 DO
      WRITE('-');
    WRITELN;
    FOR I := 0 TO LISTCNT DO
    BEGIN (1)
      IF LISTINFO[I].INLIST THEN
      BEGIN (2)
        WRITELN(' ',LISTINFO[I].NAME);
        COUNT := COUNT + 1;
      END;  (2)
    END;  (1)
    FOR I := 1 TO 40 DO
      WRITE('-');
    GOTOXY(50,2);
    WRITE('INSTRUCTIONS');
    GOTOXY(45,4);
    WRITE('To add a subtest to the list,');
    GOTOXY(45,5);
    WRITE('press the letter ''A''.  You');
    GOTOXY(45,6);
    WRITE('may have up to 20 subtests.');
    GOTOXY(45,8);
    WRITE('To remove a subtest from the');
    GOTOXY(45,9);
    WRITE('list, press <RET> until the');
    GOTOXY(45,10);
    WRITE('cursor is next to the subtest');
    GOTOXY(45,11);
    WRITE('you wish to remove, then press');
    GOTOXY(45,12);
    WRITE('the letter ''R''.');
    GOTOXY(45,14);
    WRITE('Press ''Q'' to quit.');
    GOTOXY(0,2);
    REPEAT
      COMMAND := GETCHAR(['A','R','Q',CHR(RET)],TRUE,FALSE,TRUE);
      CASE COMMAND OF
        'A' : ADDSUBTEST;
        'R' : BEGIN (3)
                I := -1;
                K := -1;
                REPEAT
                  I := I + 1;
                  IF LISTINFO[I].INLIST THEN
                    K := K + 1;
                UNTIL K = CURSORLOC;
                LISTINFO[I].INLIST := FALSE;
              END;  (3)
        'Q' : ;
      END;  (* cases *)
      IF COMMAND = CHR(RET) THEN
      BEGIN (4)
        CURSORLOC := CURSORLOC + 1;
        IF CURSORLOC > (COUNT - 1) THEN
        BEGIN (5)
          GOTOXY(0,2);
          CURSORLOC := 0;
        END  (5)
        ELSE
          GOTOXY(0,2 + CURSORLOC);
      END;  (4)
    UNTIL COMMAND <> CHR(RET);
  UNTIL (COMMAND = 'Q') AND (COUNT <= GMAXSUBTEST);
```

```
      ADJUSTLIST;
    END;  (* select 20 *)

    (* ------------------------------------------------------------------- *)

    (* display the subtest order *)
    PROCEDURE ORDER;
    TYPE VALIDTESTNUMBERS = 1..GMAXSUBTEST;
    VAR I,
        J,
        K,
        NUM   : INTEGER;
        NOTESTDUPLICATION,
        MATCH : BOOLEAN;
        BUCKET : SET OF VALIDTESTNUMBERS;
        CH : CHAR;

        (* ............................................................. *)

        (* display current order *)
        PROCEDURE SHOWTESTS;
        BEGIN
          PAGE(OUTPUT);
          K := 0;
          GOTOXY(50,3);
          WRITE('Current Order');
          FOR I := 1 TO GMAXSUBTEST DO
          BEGIN {1}
            IF SPARAMS.SUBORDER[I] <= MAXSUBTESTS THEN
            BEGIN {2}
              GOTOXY(40,I+3);
              K := K + 1;
              WRITE(' ':2,CHR(I+64),'. ');
              WRITE(' ',NAMEORDER[I]);
            END   {2}
          END;  {1}
        END;   (* showtests *)

    (* ................................................................... *)

    BEGIN
      TEMP := SPARAMS;
      SHOWTESTS;
      GOTOXY(0,0);
      WRITELN(
'Type the letter next to the subtest you wish to put in the new order.');
      WRITELN('Or press <ret> to keep the same subtest in order.');
      IF K > 0 THEN
      BEGIN {1}
        BUCKET := [];
        GOTOXY(15,3);
        WRITE('New Order');
        FOR I := 1 TO K DO
        BEGIN {2}
          NO_TEST_DUPLICATION := FALSE;
          REPEAT
            GOTOXY(2,I+3);
            WRITE(I,'.  ');
            CH := GETCHAR(['A'..CHR(K+64),CHR(13)],TRUE,FALSE,
                           TRUE);
            IF CH <> CHR(13) THEN
              NUM := ORD(CH) - 64
            ELSE
            IF CH = CHR(13) THEN
            BEGIN {3}
              NUM := I;
              SPARAMS.SUBORDER[I] := NUM;
              GOTOXY(40,I+3);
              WRITE(' ':39);
              GOTOXY(6,I+3);
              WRITE(' ':32);
              GOTOXY(6,I+3);
              WRITELN(NAMEORDER[NUM]);
            END;   (3)
            IF NUM IN BUCKET THEN
            BEGIN (4)
              GOTOXY(6,I+3);
              WRITELN('test duplication              ');
              SQUAWK;
            END    (4)
            ELSE
            BEGIN (5)
              BUCKET := BUCKET + [NUM];
              NO_TEST_DUPLICATION := TRUE;
              GOTOXY(40,NUM+3);
              WRITE(' ':39);
              GOTOXY(6,I+3);
              WRITE(' ':32);
              GOTOXY(6,I+3);
              WRITELN(NAMEORDER[NUM]);
              SPARAMS.SUBORDER[I] := NUM;
            END;   (5)
          UNTIL NO_TEST_DUPLICATION;
        END;   (2)
        FOR I := 1 TO K DO
        BEGIN (6)
          J := SPARAMS.SUBORDER[I];
          TEMP.SUBSTRAT[I] := SPARAMS.SUBSTRAT[J];
          TEMP.SUBSTOP[I] := SPARAMS.SUBSTOP[J];
          TEMP.SUBLENGTH[I] := SPARAMS.SUBLENGTH[J];
          TEMP.CKERROR[I] := SPARAMS.CKERROR[J];
          TEMPNAMEORDER[I] := NAMEORDER[J];
          TEMP.SUBORDER[I] := I;
        END;   (6)
        FOR I := (K+1) TO GMAXSUBTEST DO
          TEMP.SUBORDER[I] := 128;     (* mark as unused *)
        SPARAMS := TEMP;
        NAMEORDER := TEMPNAMEORDER;
      END;   (1)
    END;     (* order *)

    (* ------------------------------------------------------------------------ *)

    (* set up subtest strategy *)
    PROCEDURE STRATEGY;
    VAR I,
        K : INTEGER;
        CH : CHAR;
        DONE : BOOLEAN;
    BEGIN
      PAGE(OUTPUT);
WRITELN('Beginning at the top of the list, type <return> to keep old strategy');
WRITELN('or type the number of the strategy you want.  Type ''Q'' to quit.');
      GOTOXY(0,3);
WRITE(
'                  Subtests        Strategy              Available Strategies');
      GOTOXY(56,5);
      WRITE('0.  NONE');
      GOTOXY(56,6);
      WRITE('1.  B 10.2.2.2.2');
      GOTOXY(56,7);
      WRITE('2.  B 5.4.3.2.1');
      GOTOXY(56,8);
      WRITE('3.  B 10.8.6.4.2');
      GOTOXY(56,9);
      WRITE('4.  TIMED');
      K := 0;
      FOR I := 1 TO GMAXSUBTEST DO
      BEGIN (1)
        IF SPARAMS.SUBORDER[I] <= MAXSUBTESTS THEN
        BEGIN (2)
          GUTOXY(0,I+3);
          WRITE(NAMEORDER[I] : 30);
          GOTOXY(31,I+3);
          WRITE('    ',SPARAMS.SUBSTRAT[I],'.');
          CASE SPARAMS.SUBSTRAT[I] OF
```

```
            0 : WRITE('None    ');
            1 : WRITE('B 102222');
            2 : WRITE('B 54321 ');
            3 : WRITE('B 108642');
            4 : WRITE('Timed   ');
          END; (* cases *)
          K := K + 1;
        END;  (2)
      END;  (1)
      I := 1;
      DONE := FALSE;
      WHILE (K > 0) AND (NOT DONE) DO
      BEGIN (3)
        GOTOXY(34,I+3);
        CH:=GETCHAR(['0'..'4',CHR(13),'Q'],TRUE,FALSE,TRUE);
        IF CH = 'Q' THEN DONE := TRUE;
        IF CH IN ['0'..'4'] THEN
        BEGIN (4)
          SPARAMS.SUBSTRAT[I] := ORD(CH) - 48;
          GOTOXY(34,I+3);
          WRITE(ORD(CH)-48,'.');
          CASE CH OF
            '0' : WRITE('None    ');
            '1' : WRITE('B 102222');
            '2' : WRITE('B 54321 ');
            '3' : WRITE('B 108642');
            '4' : WRITE('Timed   ');
          END;  (* cases *)
        END;  (4)
        I := I + 1;
        IF I > K THEN
          I := 1;
      END;  (3)
    END;      (* strategy *)

    (* ---------------------------------------------------------------------- *)

    (* set the subtest stop flags *)
    PROCEDURE STOP;
    VAR I,
        TCNT : INTEGER;
        CH : CHAR;
        DONE : BOOLEAN;
    BEGIN
      PAGE(OUTPUT);
WRITELN('Beginning at the top of the list, press <return> to keep old value');
      WRITELN('or press ''Y'' or ''N''.  Press ''Q'' to quit.');
      GOTOXY(24,3);
      WRITE('Subtests                     Stop');
      TCNT := 0;
      FOR I := 1 TO GMAXSUBTEST DO
      BEGIN (1)
        IF SPARAMS.SUBORDER[I] <= MAXSUBTESTS THEN
        BEGIN (2)
          GOTOXY(6,I+3);
          WRITE(NAMEORDER[I] : 30);
          GOTOXY(54,I+3);
          IF SPARAMS.SUBSTOP[I] THEN
            WRITE('Y')
          ELSE
            WRITE('N');
          TCNT := TCNT + 1;
        END;  (2)
      END;  (1)
      DONE := FALSE;
      I := 1;
      WHILE (TCNT > 0) AND (NOT DONE) DO
      BEGIN (3)
        GOTOXY(54,I+3);
        CH := GETCHAR(['Q','N','Y',CHR(13)],TRUE, (* ADD y AND n HERE *)
                      TRUE,TRUE);
        IF CH = 'Y' THEN
          SPARAMS.SUBSTOP[I] := TRUE
    ELSE
      IF CH = 'N' THEN
        SPARAMS.SUBSTOP[I] := FALSE
      ELSE
        IF CH = 'Q' THEN
          DONE := TRUE;
    I := I + 1;
    IF I > TCNT THEN
      I := 1;
  END;  (3)
END;    (* stop *)

(* ---------------------------------------------------------------------- *)

(* set up # of questions/subtest *)
PROCEDURE MAXQUESTIONS;
VAR I,
    TCNT :  INTEGER;
    CH : CHAR;
    CHVALUE : PACKED ARRAY[0..1] OF CHAR;
    DONE,
    OK : BOOLEAN;

    (* .................................................................. *)

    (* read in a string and save in a temporary buffer  *)
    PROCEDURE MAXFILLBUF(CHARCNT : INTEGER;
                         OKSET : SETOFCHAR; ERASE : BOOLEAN);
    VAR I : INTEGER;
        IDCHAR : CHAR;
    BEGIN
      I := 1;
      REPEAT
        IF I > (CHARCNT-1) THEN
          IDCHAR :=
            GETCHAR([CHR(LARROW),CHR(RET)],TRUE,TRUE,TRUE)
        ELSE
        BEGIN (1)
          IDCHAR :=
            GETCHAR(OKSET + [CHR(RET),
                    CHR(LARROW),CHR(RARROW)],
                    TRUE,TRUE,TRUE);
          IF IDCHAR IN OKSET THEN
          BEGIN (2)
            LINEBUF[I] := IDCHAR;
            I := I + 1;
          END;  (2)
        END;  (1)
        IF IDCHAR = CHR(LARROW) THEN
        BEGIN (3)
          IF I = 0 THEN
            SQUAWK
          ELSE
          BEGIN (4)
            WRITE(CHR(LARROW));
            I := I - 1;
            IF ERASE THEN
            BEGIN (5)
              WRITE(' ');
              WRITE(CHR(LARROW));
              LINEBUF[I] := ' ';
            END;  (5)
          END;  (4)
        END   (3)
        ELSE
          IF IDCHAR = CHR(RARROW) THEN
          BEGIN (6)
            WRITE(LINEBUF[I]);
            I := I + 1;
          END;  (6)
      UNTIL IDCHAR = CHR(RET);
    END;  (* maxfillbuf *)
    (* ............................................................................ *)

    BEGIN  (* maxquestions *)
      PAGE(OUTPUT);
WRITELN('From top of list, press <RET> to keep old value.  To change length,');
WRITELN('type the new value you want, then press <RET>. Press ''Q'' to quit.');
      GOTOXY(24,3);
      WRITELN('Subtests                # of questions');
      TCNT := 0;
      FOR I := 1 TO GMAXSUBTEST DO
      BEGIN {1}
        IF SPARAMS.SUBORDER[I] <= MAXSUBTESTS THEN
        BEGIN {2}
          GOTOXY(6,I+3);
          WRITE(NAMEORDER[I] : 30);
          GOTOXY(55,I+3);
          WRITE(SPARAMS.SUBLENGTH[I]);
          TCNT := TCNT + 1;
        END;  {2}
      END;  {1}
      I := 1;
      DONE := FALSE;
      WHILE (TCNT > 0) AND (NOT DONE) DO
      BEGIN {3}
        GOTOXY(55,I+3);
        CH := GETCHAR(['Q',CHR(13)] + ['0'..'9'],TRUE,
                       TRUE,TRUE);
        IF CH = 'Q' THEN
          DONE := TRUE
        ELSE
          IF CH <> CHR(13) THEN
          BEGIN {4}
            OK := FALSE;
            REPEAT
              FILLCHAR(LINEBUF[0],2,' ');
              LINEBUF[0] := CH;
              GOTOXY(55,I+3);
              WRITE('  ');
              GOTOXY(55,I+3);
              WRITE(CH);
              MAXFILLBUF(2,['0'..'9'],TRUE);
              MOVELEFT(LINEBUF[0],CHVALUE[0],2);
              IF CHVALUE = '  ' THEN
                SQUAWK
              ELSE
              BEGIN {5}
                IF LINEBUF[1] = ' ' THEN
                  SPARAMS.SUBLENGTH[I] := ORD(LINEBUF[0]) - 48
                ELSE
                  SPARAMS.SUBLENGTH[I] := ((ORD(LINEBUF[0]) - 48) * 10)
                    + (ORD(LINEBUF[1]) - 48);
              END;  {5}
              IF (SPARAMS.SUBLENGTH[I] <= 0) OR (SPARAMS.SUBLENGTH[I] >20)
              THEN
                SQUAWK
              ELSE
                OK := TRUE;
            UNTIL OK;
          END;  {4}
        I := I + 1;
        IF I > TCNT THEN
          I := 1;
      END;  {3}
    END;   (* maxquestions *)

    (* ------------------------------------------------------------------------- *)

    (* set ckerror value *)
    PROCEDURE CHECKERROR;
    VAR I,
        TCNT,
        K : INTEGER;
        CH : CHAR;
        DONE,
        OK : BOOLEAN;
        CHVALUE : PACKED ARRAY[0..3] OF CHAR;
    BEGIN
      PAGE(OUTPUT);
WRITELN('Beginning at the top of the list, type <return> to keep the old');
WRITELN('value or type in the value you want.  Type ''Q'' to quit.');
      GOTOXY(24,3);
      WRITE('Subtests                Check error value');
      TCNT := 0;
      FOR I := 1 TO GMAXSUBTEST DO
      BEGIN {1}
        IF SPARAMS.SUBORDER[I] <= MAXSUBTESTS THEN
        BEGIN {2}
          GOTOXY(6,I+3);
          WRITE(NAMEORDER[I] : 30);
          GOTOXY(54,I+3);
          WRITE(SPARAMS.CKERROR[I] : 5 : 3);
          TCNT := TCNT + 1;
        END;  {2}
      END;  {1}
      I := 1;
      DONE := FALSE;
      WHILE (TCNT > 0) AND (NOT DONE) DO
      BEGIN {3}
        OK := FALSE;
        REPEAT
          GOTOXY(55,I+3);
          CH := GETCHAR(['Q','0'..'9',CHR(13)],
                        TRUE,TRUE,TRUE);
          IF CH = 'Q' THEN
            DONE := TRUE
          ELSE
            IF CH <> CHR(13) THEN
            BEGIN {4}
              GOTOXY(56,I+3);
              WRITE('.    ');
              CHVALUE[0] := CH;
              GOTOXY(57,I+3);
              K := 0;
              REPEAT
                K := K + 1;
                CH := GETCHAR(['0'..'9',CHR(27)],TRUE,TRUE,TRUE);
                CHVALUE[K] := CH;
              UNTIL (K = 3) OR (CH = CHR(27));
              IF CH <> CHR(27) THEN
                OK := TRUE
              ELSE
              BEGIN {5}
                GOTOXY(55,I+3);
                WRITE(SPARAMS.CKERROR[I] : 5 : 3);
              END;  {5}
              IF OK THEN
                SPARAMS.CKERROR[I] :=
                  (ORD(CHVALUE[0]) - 48) +
                  ((ORD(CHVALUE[1]) - 48) * 0.1) +
                  ((ORD(CHVALUE[2]) - 48) * 0.01) +
                  ((ORD(CHVALUE[3]) - 48) * 0.001);
            END;  {4}
        UNTIL (DONE) OR (OK) OR (CH = CHR(13));
        I := I + 1;
        IF I > TCNT THEN
          I := 1;
      END;  {3}
    END;  (* check error *)

    (* ---------------------------------------------------------------------- *)

    (* parameter setup menu *)
    PROCEDURE PARAMMENU;
    VAR X,Y : INTEGER;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(19,0);
      WRITE('MODIFY PARAMETERS MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following procedures by entering its number.');
      X := 16;
      Y := 8;
      GOTOXY(X,Y);
      WRITE('1.  QUIT');
      GOTOXY(X,Y+1);
      WRITE('2.  MODIFY LIST OF SUBTESTS TO GIVE');
      GOTOXY(X,Y+2);
      WRITE('3.  ORDER THE SUBTESTS');
      GOTOXY(X,Y+3);
      WRITE('4.  CONFIGURE SUBTEST LENGTH');
      GOTOXY(X,Y+4);
      WRITE('5.  CONFIGURE SUBTEST STRATEGY');
      GOTOXY(X,Y+5);
      WRITE('6.  CONFIGURE VARIANCE CUT-OFF');
      GOTOXY(X,Y+6);
      WRITE('7.  CONFIGURE SUBTEST STOP FLAG');
      GOTOXY(X,Y+10);
      WRITE('Enter Choice # : ');
    END;  (* parammenu *)

    (* ------------------------------------------------------------------------- *)

    (* stores in the sparams.suborder array, the record numbers of the *)
    (* subtest directories so we won't have to search the test         *)
    (* directory later to find the subtest directory we want.          *)
    PROCEDURE SETSUBTESTRECNUMS;
    VAR K,
        RECNUM : INTEGER;  (* # of key errors in samples questions *)
        FOUND    : BOOLEAN;
    BEGIN
      FOR K := 1 TO GMAXSUBTEST DO
      BEGIN {1}
        IF SPARAMS.SUBORDER[K] <= MAXSUBTESTS THEN
        BEGIN {2}
           RECNUM := 0;
           (* search through test directory sequentially *)
           FOUND := FALSE;
           REPEAT
             IF (NOT (DIRINFO[RECNUM].NOTUSED))
                AND (DIRINFO[RECNUM].TNAME = NAMEORDER[K]) THEN
               FOUND := TRUE
             ELSE
               RECNUM := RECNUM + 1;
           UNTIL (FOUND) OR (RECNUM > MAXSUBTESTS);

           IF FOUND THEN
             SPARAMS.SUBORDER[K] := RECNUM
           ELSE
           BEGIN {3}
             PAGE(OUTPUT);
             WRITELN('Subtest configuration error !');
             WRITELN('Test ',NAMEORDER[K],' in list, but not in directory !');
             SQUAWK;
             WRITELN;
             STALL;
           END;  {3}
        END;  {2}
      END;  {1}
    END;  (*  setsubtestrecnums *)

(* ............................................................................ *)

BEGIN   (* Set parameters *)
  REPEAT
    PARAMMENU;
    COMMAND := GETCHAR(['1'..'7'],TRUE,FALSE,TRUE);
    CASE COMMAND OF
      '1' : ;
      '2' : SELECT20;
```

```
        '3' : ORDER;
        '4' : MAXQUESTIONS;
        '5' : STRATEGY;
        '6' : CHECKERROR;
        '7' : STOP;
      END;
  UNTIL COMMAND = '1';
  SETSUBTESTRECNUMS;
END;     (* set parameters *)
```

```
(*************************************************************************)
(*                                                                       *)
(*      Textfile : PMGR.DIR/P.FBACK.TEXT          Volume : TFILES        *)
(*      Codefile : P.MGR.CODE ('Include' file)    Volume : CATDATA       *)
(*                                                                       *)
(*************************************************************************)
(*             DEC.  1, 1982                      NPROC                  *)
(*************************************************************************)


(* configure feedback *)
PROCEDURE SETFBACK;
VAR COMMAND,
   CH  : CHAR;

    (* ................................................................. *)

    (* feedback configure menu *)
    PROCEDURE FMENU;
    VAR X,Y : INTEGER;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(18,0);
      WRITE('CONFIGURE FEEDBACK MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following procedures by entering its number.');
      X := 18;
      Y := 8;
      GOTOXY(X,Y);
      WRITE('1.  QUIT');
      GOTOXY(X,Y+1);
      WRITE('2.  CONFIGURE QUESTION FEEDBACK');
      GOTOXY(X,Y+2);
      WRITE('3.  CONFIGURE SUBTEST FEEDBACK');
      GOTOXY(X,Y+3);
      WRITE('4.  CONFIGURE SESSION FEEDBACK');
      GOTOXY(X,Y+7);
      WRITE('Enter Choice # : ');
    END;   (* fmenu *)

    (* ----------------------------------------------------------------- *)

    (* configure item feedback/output *)
    PROCEDURE ITEM;
    VAR X,Y,OPT : INTEGER;
        CHAROPT : CHAR;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(18,0);
      WRITE('QUESTION FEEDBACK MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following options by entering its number.');
      X := 16;
      Y := 9;
      GOTOXY(X,Y);
      WRITE('1.  QUIT');
      GOTOXY(X,Y+1);
      WRITE('2.  NO QUESTION FEEDBACK');
      GOTOXY(X,Y+2);
      WRITE('3.  RIGHT/WRONG QUESTION FEEDBACK');
      GOTOXY(X,Y+3);
      WRITE('4.  REMEDIAL QUESTION FEEDBACK');
      REPEAT
        GOTOXY(0,6);
        WRITE('You currently have option : ');
        IF SPARAMS.ITEMOUTPUT = 0 THEN
          OPT := 2
        ELSE
          IF SPARAMS.ITEMFB = 1 THEN
            OPT := 3
          ELSE
            OPT := 4;
        WRITE(OPT);
```

133

```
    INVERSE;
    CASE OPT OF
      2 : BEGIN
            GOTOXY(X,Y+1);
            WRITE('2.  NO QUESTION FEEDBACK');
          END;
      3 : BEGIN
            GOTOXY(X,Y+2);
            WRITE('3.  RIGHT/WRONG QUESTION FEEDBACK');
          END;
      4 : BEGIN
            GOTOXY(X,Y+3);
            WRITE('4.  REMEDIAL QUESTION FEEDBACK');
          END;
    END; (* cases *)
    NORMAL;
    GOTOXY(X,Y+7);
    WRITE('                          ');
    GOTOXY(X,Y+7);
    WRITE('Enter Choice # : ');
    CHAROPT := GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
    CASE CHAROPT OF
      '2' : SPARAMS.ITEMOUTPUT := 0;
      '3' : BEGIN
              SPARAMS.ITEMOUTPUT := 1;
              SPARAMS.ITEMFB := 1;
            END;
      '4' : BEGIN
              SPARAMS.ITEMOUTPUT := 1;
              SPARAMS.ITEMFB := 2;
            END;
    END;  (* cases *)

    CASE OPT OF
      2 : BEGIN
            GOTOXY(X,Y+1);
            WRITE('2.  NO QUESTION FEEDBACK');
          END;
      3 : BEGIN
            GOTOXY(X,Y+2);
            WRITE('3.  RIGHT/WRONG QUESTION FEEDBACK');
          END;
      4 : BEGIN
            GOTOXY(X,Y+3);
            WRITE('4.  REMEDIAL QUESTION FEEDBACK');
          END;
    END;  (* cases *)
  UNTIL CHAROPT = '1';
END;      (* item *)

(* -------------------------------------------------------------------------- *)

(* configure subtest feedback/output *)
PROCEDURE SUBTEST;
VAR X,Y,OPT : INTEGER;
    CHAROPT : CHAR;
BEGIN
  PAGE(OUTPUT);
  GOTOXY(18,0);
  WRITE('SUBTEST FEEDBACK MENU');
  GOTOXY(0,4);
  WRITE('Select one of the following options by entering its number.');
  X := 16;
  Y := 9;
  GOTOXY(X,Y);
  WRITE('1.  QUIT');
  GOTOXY(X,Y+1);
  WRITE('2.  NO SUBTEST FEEDBACK');
  GOTOXY(X,Y+2);
  WRITE('3.  SUBTEST FEEDBACK TO SCREEN');
  GOTOXY(X,Y+3);
  WRITE('4.  SUBTEST FEEDBACK TO PRINTER');
  GOTOXY(X,Y+4);
  WRITE('5.  SUBTEST FEEDBACK TO SCREEN AND PRINTER');
  REPEAT
    GOTOXY(0,6);
    WRITE('You currently have option : ');
    IF SPARAMS.SUBTESTOUTPUT = 0 THEN
      OPT := 2
    ELSE
      IF SPARAMS.SUBTESTOUTPUT = 1 THEN
        OPT := 3
      ELSE
        IF SPARAMS.SUBTESTOUTPUT = 2 THEN
          OPT := 4
        ELSE
          OPT := 5;
    WRITE(OPT);

    INVERSE;
    CASE OPT OF
      2 : BEGIN
            GOTOXY(X,Y+1);
            WRITE('2.  NO SUBTEST FEEDBACK');
          END;
      3 : BEGIN
            GOTOXY(X,Y+2);
            WRITE('3.  SUBTEST FEEDBACK TO SCREEN');
          END;
      4 : BEGIN
            GOTOXY(X,Y+3);
            WRITE('4.  SUBTEST FEEDBACK TO PRINTER');
          END;
      5 : BEGIN
            GOTOXY(X,Y+4);
            WRITE('5.  SUBTEST FEEDBACK TO SCREEN AND PRINTER');
          END;
    END;  (* cases *)
    NORMAL;

    GOTOXY(X,Y+8);
    WRITE('                    ');
    GOTOXY(X,Y+8);
    WRITE('Enter Choice # : ');
    CHAROPT := GETCHAR(['1'..'5'],TRUE,TRUE,TRUE);
    CASE CHAROPT OF
      '2' : SPARAMS.SUBTESTOUTPUT := 0;
      '3' : SPARAMS.SUBTESTOUTPUT := 1;
      '4' : SPARAMS.SUBTESTOUTPUT := 2;
      '5' : SPARAMS.SUBTESTOUTPUT := 3;
    END;  (* cases *)

    CASE OPT OF
      2 : BEGIN
            GOTOXY(X,Y+1);
            WRITE('2.  NO SUBTEST FEEDBACK');
          END;
      3 : BEGIN
            GOTOXY(X,Y+2);
            WRITE('3.  SUBTEST FEEDBACK TO SCREEN');
          END;
      4 : BEGIN
            GOTOXY(X,Y+3);
            WRITE('4.  SUBTEST FEEDBACK TO PRINTER');
          END;
      5 : BEGIN
            GOTOXY(X,Y+4);
            WRITE('5.  SUBTEST FEEDBACK TO SCREEN AND PRINTER');
          END;
    END;  (* cases *)
  UNTIL CHAROPT = '1';
END;      (* subtest *)

(* ---------------------------------------------------------------------- *)
(* configure session feedback/output *)
PROCEDURE SESSION;
VAR X,Y,OPT : INTEGER;
    CHAROPT : CHAR;
BEGIN
  PAGE(OUTPUT);
  GOTOXY(18,0);
  WRITE('SESSION FEEDBACK MENU');
  GOTOXY(0,4);
  WRITE('Select one of the following options by entering its number.');
  X := 16;
  Y := 9;
  GOTOXY(X,Y);
  WRITE('1.  QUIT');
  GOTOXY(X,Y+1);
  WRITE('2.  NO SESSION FEEDBACK');
  GOTOXY(X,Y+2);
  WRITE('3.  SESSION FEEDBACK TO SCREEN');
  GOTOXY(X,Y+3);
  WRITE('4.  SESSION FEEDBACK TO PRINTER');
  GOTOXY(X,Y+4);
  WRITE('5.  SESSION FEEDBACK TO SCREEN AND PRINTER');
  REPEAT
    GOTOXY(0,6);
    WRITE('You currently have option : ');
    IF SPARAMS.SESSIONOUTPUT = 0 THEN
      OPT := 2
    ELSE
      IF SPARAMS.SESSIONOUTPUT = 1 THEN
        OPT := 3
      ELSE
        IF SPARAMS.SESSIONOUTPUT = 2 THEN
          OPT := 4
        ELSE
          OPT := 5;
    WRITE(OPT);

    INVERSE;
    CASE OPT OF
      2 : BEGIN
            GOTOXY(X,Y+1);
            WRITE('2.  NO SESSION FEEDBACK');
          END;
      3 : BEGIN
            GOTOXY(X,Y+2);
            WRITE('3.  SESSION FEEDBACK TO SCREEN');
          END;
      4 : BEGIN
            GOTOXY(X,Y+3);
            WRITE('4.  SESSION FEEDBACK TO PRINTER');
          END;
      5 : BEGIN
            GOTOXY(X,Y+4);
            WRITE('5.  SESSION FEEDBACK TO SCREEN AND PRINTER');
          END;
    END;  (* cases *)
    NORMAL;

    GOTOXY(X,Y+8);
    WRITE('                         ');
    GOTOXY(X,Y+8);
    WRITE('Enter Choice # : ');
    CHAROPT := GETCHAR(['1'..'5'],TRUE,TRUE,TRUE);
    CASE CHAROPT OF
    '2' : SPARAMS.SESSIONOUTPUT := 0;
    '3' : SPARAMS.SESSIONOUTPUT := 1;
    '4' : SPARAMS.SESSIONOUTPUT := 2;
    '5' : SPARAMS.SESSIONOUTPUT := 3;
    END;  (* cases *)

    CASE OPT OF
      2 : BEGIN
            GOTOXY(X,Y+1);
```

136

```
                WRITE('2.  NO SESSION FEEDBACK');
              END;
          3 : BEGIN
                GOTOXY(X,Y+2);
                WRITE('3.  SESSION FEEDBACK TO SCREEN');
              END;
          4 : BEGIN
                GOTOXY(X,Y+3);
                WRITE('4.  SESSION FEEDBACK TO PRINTER');
              END;
          5 : BEGIN
                GOTOXY(X,Y+4);
                WRITE('5.  SESSION FEEDBACK TO SCREEN AND PRINTER');
              END;
        END;  (* cases *)
      UNTIL CHAROPT = '1';
    END;     (* session *)

(* ........................................................................ *)

BEGIN  (* setfback *)
  REPEAT
    FMENU;
    COMMAND := GETCHAR(['1'..'4'],TRUE,FALSE,TRUE);
    CASE COMMAND OF
      '1' : ;
      '2' : ITEM;
      '3' : SUBTEST;
      '4' : SESSION;
    END;  (* cases *)
  UNTIL COMMAND = '1';
END;   (* setfback *)
```

TMGR.DIR:
Subdirectory - Test Manager Textfiles

```
(*$S+*)
(******************************************************************************)
(*                                                                            *)
(*      Textfile : TMGR.DIR/T.MGR.TEXT            Volume : TFILES             *)
(*      Codefile : T.MGR.CODE                     Volume : CATDATA            *)
(*                                                                            *)
(******************************************************************************)
(*  File last modified : Feb 28, 1983             NPROC                       *)
(******************************************************************************)
(*                                                                            *)
(*  Functions : 1) provides file maintenance of tests.                        *)
(*              2) allows creation of new tests.                              *)
(*              3) access to existing tests for insertion, modification, or   *)
(*                 deletion of test questions.                                *)
(*              4) lists tests to file, console, or printer for verification. *)
(*              5) writes subtests from Corvus to five inch floppy.           *)
(*                                                                            *)
(******************************************************************************)
(*                                                                            *)
(* 1. Slot 0 of subtest directory reserved for subtest instructions.          *)
(*                                                                            *)
(* 2. Slots 1..5 of subtest directory reserved for subtest samples.           *)
(*                                                                            *)
(* 3. Variant type of answers stored.  May store item answer as character,    *)
(*    integer, or packed array of 7 characters.  This handles the different   *)
(*    types of subtests.                                                      *)
(*                                                                            *)
(* 4. First character of subtest name specifies type of screen format.        *)
(*    * = 80 char/inverse, @ = 80 char/normal, ? = 40 char/normal.  If none   *)
(*    of these characters occur, then the screen defaults to 40 char/inverse. *)
(*                                                                            *)
(* 5. If test is a timed test, the second character of the subtest name,      *)
(*    will be a digit and specify the maximum time in minutes, allowed for    *)
(*    that subtest.                                                           *)
(*                                                                            *)
(* 6. Graphics feature added, if graphics flag is set the program will look   *)
(*    for a graphics file to display, else it looks in the normal ascii file. *)
(*                                                                            *)
(* 7. Text output routine improved. Text format slightly changed.  After      *)
(*    gotoflag and x and y coordinates, the number of bytes for that line is  *)
(*    saved.  Text output consists of mass movement of bytes into line        *)
(*    buffer, then writing the buffer to the screen for each line.            *)
(*                                                                            *)
(* 8. Timed subtests now can be set to minutes and seconds.  Two more         *)
(*    characters in subtest name used for seconds.                            *)
(*                                                                            *)
(* 9. Will display compressed graphics files as well as normal fotofiles.     *)
(*                                                                            *)
(******************************************************************************)

PROGRAM TESTMGR;
USES PGRAF,
     CHAINSTUFF;

CONST (* ascii values *)
      ETX = 3;          (* control-c *)
      BELL = 7;
      NUL = 0;
      LARROW = 8;
      RARROW = 21;
      RET = 13;
      UP = 11;          (* up arrow *)
      DOWN = 10;        (* down arrow *)
      ESC = 27;
      SPACE = 32;
      PAGEOUT = 16;  (* cntrl-p on apple ii *)
      ASCIIOFFSET = 48; (* ascii zero *)
      NIL = -1;

      XSCREEN = 79;  (* screen dimensions *)
      YSCREEN = 23;

      (* block sized buffer for ascii *)
```

```
        MAXITEMBUF = 2047;

        MAXLINEBUF = 79;    (* string buffer size *)

        (* screen boundaries for question text *)
        (* lines 0 - 3, 20 - 23 reserved for   *)
        (* system messages.                    *)
        TOPMAX = 0;
        LEFTMAX = 0;
        BOTTOMMAX = 19;

        (* question textfile control codes *)
        GOTOFLAG = 128;       (* flags a gotoxy *)
        PAGEFLAG = 129;       (* flags text continues on another page *)
        UNUSEDFLAG = 130;     (* flags unused byte *)
        ENDITEM = 131;        (* flags end of text for a question *)

        (* these files must reside on disk ! *)
        DATANAME = 'CATDATA:ITEMPOOL.DATA';    (* question data *)
        TEXTNAME = 'QTEXT:ITEMTEXT.DATA';    (* question text *)
        INDEXNAME = 'CATDATA:TESTINDEX.DATA'; (* test directory *)

        (* output file for listing test questions *)
        DEFAULTFILE = 'CATDATA:TEST.TEXT';

        (* slots available in directory *)
        MAXSUBTESTS = 20;
        (* maximum question pool per test *)
        MAXITEMPOOL = 300;

        (* maximum # of sample questions allowed *)
        MAXSAMPLES = 5;

        (* printer file name *)
        UNITNUMPRINTER = 'PRINTER:';

        VERSION = '[1.03]';

        (* flag for compressed graphics *)
        COMPRESSED = 1.0;


TYPE DIRDATA = PACKED RECORD  (* directory for tests *)
                      UNUSED : BOOLEAN;   (* tells if record occupied *)
                      TESTNAME : STRING;  (* name of subtest *)
                      (* subtest directory of question id codes *)
                      ITEMCODE : PACKED ARRAY
                                 [0..MAXITEMPOOL]
                                 OF INTEGER;
                   END;

       (* types of answers to questions *)
       ITEMRESPONSES = (CHARVALUE,INTVALUE,SEVENCHR);

       ITEMDATA = PACKED RECORD  (* question ptrs/data *)

                      (* flag this as graphics item *)
                      GRAPHICS : BOOLEAN;
                      (* bounds for response range *)
                      LOWANSWER,
                      HIGHANSWER : CHAR;
                      (* block # where question text starts *)
                      ITEMBLOCK,
                      (* byte ptr where question text starts *)
                      ITEMPTR,
                      (* # of answers for multiple question screens *)
                      ANSWERCOUNT : INTEGER;
                      (* information parameters *)
                      A,B,C,

                      (* currently unused *)
                      PROPCORRECT,
                        POINTBISERIAL,
                        YOPT,
                        XOPT,
                        DUMMY1,
                        DUMMY2,
                        DUMMY3  : REAL;

                        (* answer to question *)
                        CASE ATYPE : ITEMRESPONSES OF
                          CHARVALUE : (ANSWER : CHAR);
                          INTVALUE  : (INTANSWER : INTEGER);
                          SEVENCHR  : (CHRANSWER : PACKED ARRAY[1..7] OF
                                                   CHAR);
                      END;

     SETOFCHAR = SET OF CHAR;


VAR LETTERS,DIGITS,CHARACTERS : SET OF CHAR;
    OUTPUT,COMMAND : CHAR;

    (* screen character buffer, this is the page buffer for item editor *)
    SCREEN : PACKED ARRAY
             [0..XSCREEN,0..YSCREEN] OF CHAR;

    (* block of ascii, control codes *)
    TRIX : RECORD CASE INTEGER OF
             1 : (ITEMBUF : ARRAY[0..1023] OF INTEGER);
             2 : (ASCIIBUF : PACKED ARRAY[0..MAXITEMBUF] OF 0..139);
           END;

    (* string character buffer *)
    LINEBUF : PACKED ARRAY[0..MAXLINEBUF] OF CHAR;

    RIGHTMAX,                (* boundary for right margin of question text *)
    PAGENUM,                 (* current page of question text *)
    CURRINDEXRECNUM,         (* record # of file directory *)
    CURRBLOCK,               (* block # of start of item text *)
    CURRFREEPTR : INTEGER;   (* ptr to free loc in block *)

    CGRAF,               (* true ==> picture is compressed graphics *)
    ESCPROC,             (* true ==> leave current procedure *)
    SCR80,               (* true ==> 80 columns screen *)
    VNORMAL : BOOLEAN;   (* true ==> normal video *)

    AVALUE,         (* item parameters *)
    BVALUE,
    CVALUE : REAL;

    (* test directory *)
    DIRECTORY : DIRDATA;
    FILEDIRECTORY : FILE OF DIRDATA;

    (* data records *)
    ITEMINFO : ITEMDATA;
    FILEITEMINFO : FILE OF ITEMDATA;

    (* file of ascii codes, control #'s *)
    ITEMTEXT : FILE;

    (* output file for test listings *)
    DEST : TEXT;

    (* directory record information *)
    DIRINFO : ARRAY[0..MAXSUBTESTS] OF RECORD

                                        (* record occupied *)
                                        NOTUSED : BOOLEAN;

                                        (* subtest name *)
                                        TNAME : STRING;

                                        (* # items in subtest *)
```

```
                                         ITEMCOUNT : INTEGER;
                                       END;


PROCEDURE PAGE(DUMMY : CHAR); FORWARD;
PROCEDURE TOPOFFORM;  FORWARD;
PROCEDURE SQUAWK;  FORWARD;
PROCEDURE BLANKLINES(START,COUNT,ENDCURSOR : INTEGER); FORWARD;
FUNCTION GETCHAR(OKSET : SETOFCHAR;
                 FLUSHQUEUE,ECHO,BEEP : BOOLEAN) : CHAR;  FORWARD;
PROCEDURE STALL;  FORWARD;
PROCEDURE FILLBUF(CHARCNT : INTEGER;
                  OKSET : SETOFCHAR; ERASE : BOOLEAN);  FORWARD;
PROCEDURE  GETNEWFILE;  FORWARD;
PROCEDURE SCRCONTROL(I,J,K : INTEGER); FORWARD;
PROCEDURE TEXT40MODE;    FORWARD;
PROCEDURE TEXT80MODE;    FORWARD;
PROCEDURE INVERSE;       FORWARD;
PROCEDURE NORMAL;        FORWARD;
PROCEDURE WRITEITEMBLOCK(WHICHBLOCK : INTEGER);    FORWARD;
PROCEDURE READITEMBLOCK(WHICHBLOCK : INTEGER);     FORWARD;
FUNCTION SLOTSEARCH(CODE : INTEGER) : INTEGER;     FORWARD;
PROCEDURE UPDATEDIRECTORY(RECNUM : INTEGER);       FORWARD;
PROCEDURE UPDATEITEMFILE(RECNUM : INTEGER);        FORWARD;
PROCEDURE SAVEPTRS;                                FORWARD;
PROCEDURE LOADPTRS;                                FORWARD;
FUNCTION HASH(KEY : INTEGER) : INTEGER;            FORWARD;
PROCEDURE DECODEPRINT(BLOCKNUM,BLOCKPTR : INTEGER);  FORWARD;
PROCEDURE GETPARAM(PTYPE : CHAR);    FORWARD;
FUNCTION ITEMFREESLOT : INTEGER;     FORWARD;
PROCEDURE SETGFLAG;                  FORWARD;
PROCEDURE SETSCREEN;                 FORWARD;
PROCEDURE GWRITESTR(GSTR : STRING);  FORWARD;
PROCEDURE GGOTOXY(X,Y : INTEGER);    FORWARD;
PROCEDURE INITFORGRAFIX;             FORWARD;
PROCEDURE GWRITECHR(GCHR : CHAR);              FORWARD;
PROCEDURE GWRITEINT(GINT : INTEGER);           FORWARD;
PROCEDURE GWRITELN;                  FORWARD;
PROCEDURE GSTALL;                    FORWARD;
PROCEDURE GBLANKLINES(STARTBLANK,
                      BLANKTHISMANY,
                      LEAVECURSOR : INTEGER);  FORWARD;
PROCEDURE GDECODEPRINT(SUBTESTNUM,ITEMCODE : INTEGER);   FORWARD;
PROCEDURE CHANGETESTNAME;  FORWARD;


(*$I /TFILES/TMGR.DIR/T.LIST.TEXT *)      (* lists tests to output *)
(*$I /TFILES/TMGR.DIR/T.SEARCH.TEXT *)    (* searches for keywords in items *)
(*$I /TFILES/TMGR.DIR/T.1SUBRT.TEXT *)    (*  utilities *)
(*$I /TFILES/TMGR.DIR/T.INSTR.TEXT *)     (* set up subtest instructions *)
(*$I /TFILES/TMGR.DIR/T.SAMPLES.TEXT *)   (* subtest sample set up *)
(*$I /TFILES/TMGR.DIR/T.GET1.TEXT *)      (* loads test, manipulate questions *)
(*$I /TFILES/TMGR.DIR/T.GET2.TEXT *)      (* loads test, manipulate questions *)
(*$I /TFILES/TMGR.DIR/T.FLOPPY.TEXT *)    (* transfer to floppy routine *)
(*$I /TFILES/TMGR.DIR/T.NEW.TEXT *)       (* new test/questions *)
(*$I /TFILES/TMGR.DIR/T.DELETE.TEXT *)    (* delete test *)
(*$I /TFILES/TMGR.DIR/T.SUBRT.TEXT *)     (* test manager utilities *)
(*$I /TFILES/TMGR.DIR/T.1UTL.TEXT *)      (* utility procedures *)
(*$I /TFILES/TMGR.DIR/T.IO.TEXT *)        (* io routines *)

(* main program *)

BEGIN

  (* initialize character sets *)
  DIGITS := ['0'..'9'];
  LETTERS := ['A'..'Z','a'..'z'];
  CHARACTERS := [CHR(32)..CHR(126)];

  (* initialize string buffer to all blanks *)
  FILLCHAR(LINEBUF[0],MAXLINEBUF,' ');

  (* read all the subtests' directory information into a record saves direc-  *)
  (* tory information in main memory so we don't have to keep going to disk   *)
  (* to find out which subtests are in the database.                          *)
  (* file : T.SUBRT.TEXT *)
  GETDIRINFO;

  INITFORGRAFIX;

  (* loop until want to leave program *)
  REPEAT

    (* set default screen conditions *)
    SCR80 := FALSE;
    VNORMAL := FALSE;
    CGRAF := FALSE;
    ESCPROC := FALSE;

    (* set right margin for question text editor *)
    RIGHTMAX := 38;

    (* display command menu *)
    (* file : T.SUBRT.TEXT *)
    MENU;

    (* get option selected *)
    COMMAND := GETCHAR(['1'..'8'],TRUE,FALSE,TRUE);
    CASE COMMAND OF

      (* leave program *)
      '1' : ;

      (* load subtest directory and present further options for subtest *)
      (* LOADTEST is in file : T.LIST *)
      (* FETCHTEST is in file : T.GET.TEXT *)
      '2' : BEGIN
              LOADTEST('Fetch which test? : ');
              IF NOT ESCPROC THEN FETCHTEST;
            END;

      (* list all subtests in directory and how many questions in each *)
      (* file : T.LIST.TEXT *)
      '3' : LISTTESTS(TRUE);

      (* file : T.NEW.TEXT *)
      '4' : NEWTEST;

      (* change the subtest name or screen format for subtest or time speci-  *)
      (* fication for timed subtests.                                         *)
      (* file : T.1SUBRT.TEXT *)
      '5' : CHANGETESTNAME;

      (* write the subtest to a floppy disk in similar but not the same file  *)
      (* layout.  Same record layout, however.                                *)
      (* file : T.FLOPPY.TEXT *)
      '6' : TRANSFERTO5INCH;

      (* remove a subtest from the directory *)
      (* file : T.DELETE.TEXT *)
      '7' : DELETETEST;

      '8' : SEARCHTEXT;
    END;
  UNTIL COMMAND = '1';

  (* jump to cat project driver *)
  SETCHAIN('CATDATA:CATPROJECT');

END.   (* t-mgr *)
```

```
(*************************************************************************)
(*                                                                       *)
(*      Textfile : TMGR.DIR/T.1UTL.TEXT         Volume : TFILES         *)
(*      Codefile : T.MGR.CODE ('Include' file)  Volume : CATDATA        *)
(*                                                                       *)
(*************************************************************************)
(*              DEC.  1, 1982                   NPRDC                   *)
(*************************************************************************)

(* clear the screen *)
PROCEDURE PAGE;
BEGIN

  (* for apple 2 *)
  WRITE(CHR(12));

  (* for apple 3 *)
  WRITE(CHR(28));
  GOTOXY(0,0);
END;  (*  page *)

{-------------------------------------------------------------------------}

(* form feeds the printer *)
PROCEDURE TOPOFFORM;
BEGIN
  REWRITE(DEST,UNITNUMPRINTER);
  WRITE(DEST,CHR(12));
  CLOSE(DEST,NORMAL);
END;    (* top of form *)

{-------------------------------------------------------------------------}

(**** rings the bell ****)
PROCEDURE SQUAWK;
BEGIN
  WRITE(CHR(BELL));
END;  (* squawk *)

{-------------------------------------------------------------------------}

(**** blank out lines  ****)
(* given a y location to begin, # of lines to erase, which line *)
(* to leave cursor.  erases only 40 column width.              *)
PROCEDURE BLANKLINES;
VAR I : INTEGER;
BEGIN
  GOTOXY(0,START);
  FOR I := 1 TO (COUNT-1) DO
    WRITELN(' ' : 39);
  WRITE(' ':39);
  GOTOXY(0,ENDCURSOR);
END;    (* blanklines *)

{-------------------------------------------------------------------------}

(* read an acceptable character from the keyboard *)
(* given a set of acceptable characters to read.  *)
(* and flags if you want computer to flush the    *)
(* type ahead buffer, beep if a bad key is pressed*)
(* or echo the character pressed.                 *)
FUNCTION GETCHAR;
VAR MASK : PACKED ARRAY[0..0] OF CHAR;
BEGIN
  IF FLUSHQUEUE THEN UNITCLEAR(2); (* flush buffer *)
  REPEAT
    UNITREAD(2,MASK,1);
    IF BEEP AND NOT (MASK[0] IN OKSET) THEN SQUAWK;
  UNTIL MASK[0] IN OKSET;
  IF ECHO AND (MASK[0] IN [CHR(32)..CHR(126)]) THEN
    WRITE(MASK[0]);
  GETCHAR := MASK[0];
END;  (* getchar *)
```

```
{---------------------------------------------------------------------------}

(**** display a message/wait for a keystroke ****)
PROCEDURE STALL;
VAR STALLCHAR : CHAR;
BEGIN
  WRITE('Press <RET> to continue ');
  STALLCHAR :=
    GETCHAR([CHR(RET),CHR(ESC)],TRUE,FALSE,TRUE);
  IF STALLCHAR = CHR(ESC) THEN EXIT(PROGRAM);
END;  (* stall *)

{---------------------------------------------------------------------------}

(* read in a string and save in a temporary buffer  *)
(* allows control on what characters may be typed   *)
(* and the number of characters typed.              *)
PROCEDURE FILLBUF;
VAR I : INTEGER;
    IDCHAR : CHAR;
BEGIN
  I := 0;
  REPEAT
    IF I > (CHARCNT-1) THEN
      IDCHAR :=
        GETCHAR([CHR(LARROW),CHR(RET)],TRUE,TRUE,TRUE)
    ELSE
    BEGIN {1}
      IDCHAR :=
        GETCHAR(OKSET + [CHR(RET),
                CHR(LARROW),CHR(RARROW)],
                TRUE,TRUE,TRUE);
      IF IDCHAR IN OKSET THEN
      BEGIN {2}
        LINEBUF[I] := IDCHAR;
        I := I + 1;
      END;  {2}
    END;  {1}
    IF IDCHAR = CHR(LARROW) THEN
    BEGIN {3}
      IF I = 0 THEN
        SQUAWK
      ELSE
      BEGIN {4}
        WRITE(CHR(LARROW));
        I := I - 1;
        IF ERASE THEN
        BEGIN {5}
          WRITE(' ');
          WRITE(CHR(LARROW));
          LINEBUF[I] := ' ';
        END;  {5}
      END;  {4}
    END  {3}
    ELSE
      IF IDCHAR = CHR(RARROW) THEN
      BEGIN {6}
        WRITE(LINEBUF[I]);
        I := I + 1;
      END;  {6}
  UNTIL IDCHAR = CHR(RET);
END;  (* fillbuf *)

{---------------------------------------------------------------------------}

(**** open a new text file ****)
PROCEDURE  GETNEWFILE;
VAR FILENAME : STRING;
    ERRNUM : INTEGER;

  {.......................................................................}
  (**** get a legal filename ****)
  FUNCTION NAMEOK : BOOLEAN;
  VAR I : INTEGER;
  BEGIN
    IF FILENAME = ''
      THEN BEGIN {1}
             FILENAME := DEFAULTFILE;
             GOTOXY(44,0);
             WRITE(FILENAME);
           END    {1}
      ELSE IF FILENAME[1] = CHR(esc) THEN EXIT(PROGRAM);
    IF (POS('.TEXT',FILENAME) <> (LENGTH(FILENAME) - 4)) OR
       (LENGTH(FILENAME) < 6 )
       THEN FILENAME := CONCAT(FILENAME,'.TEXT');
  (*$I-*)
    RESET(DEST,FILENAME);
  (*$I+*)
    IF IORESULT = 0
      THEN BEGIN {2}
             WRITELN;
             WRITELN;
             WRITE('Destroy old ',FILENAME,'?  Press ''N'' or ''Y'' ');
             IF GETCHAR(['y','n','Y','N'],TRUE,TRUE,TRUE) IN ['Y','y']
               THEN BEGIN {3}
                      CLOSE(DEST,PURGE);
                      REWRITE(DEST,FILENAME);
                      NAMEOK := TRUE;
                    END    {3}
               ELSE BEGIN {4}
                      CLOSE(DEST,LOCK);
                      NAMEOK := FALSE;
                    END;  {4}
           END    {2}
      ELSE BEGIN {5}
           (*$I-*)
             REWRITE(DEST,FILENAME);
           (*$I+*)
             ERRNUM := IORESULT;
             IF IORESULT <> 0
              THEN BEGIN {6}
                     WRITELN;
                     WRITELN;
                     WRITELN('Cannot open ',FILENAME,' Io error #',ERRNUM);
                     NAMEOK := FALSE;
                   END    {6}
              ELSE NAMEOK := TRUE;
           END;  {5}
  END; (* nameok *)

 {...........................................................................}

BEGIN  (* getnewfile  *)
  REPEAT
    PAGE(OUTPUT);
    WRITE('Enter output file name, then press <RET>  : ');
    READLN(FILENAME);
  UNTIL NAMEOK;
END;

 {---------------------------------------------------------------------------}

(* send control characters to screen *)
PROCEDURE SCRCONTROL;          { PASCAL interface to Screen Control}
VAR N: INTEGER;                        { APPLE III Standard Device Drivers}
    G_ARRAY:PACKED ARRAY[0.. 3] OF 0..255;       {....... Pages 34 to 46.}
BEGIN
  G_ARRAY[0]:= I; G_ARRAY[1]:=J; G_ARRAY[2]:=K;
  UNITWRITE(1,G_ARRAY,3,,12);
END;    (* scrcontrol *)

 {---------------------------------------------------------------------------}

(* switch to 40 column screen *)
```

GNL

```
PROCEDURE TEXT40MODE;
BEGIN
  SCRCONTROL(16,0,28);       {Text mode 40BW, followed by clearscreen.}
END;  (* text40mode *)

{-----------------------------------------------------------------------------}

(* switch to 80 column screen *)
PROCEDURE TEXT80MODE;
BEGIN
  SCRCONTROL(04,0,0);       {Restore Viewport to its original condition.}
  SCRCONTROL(16,2,28);      {Text Mode 80, followed by clearscreen.}
END;    (* text80mode *)

{-----------------------------------------------------------------------------}

(* turn on reverse video *)
(* black on white        *)
PROCEDURE INVERSE;
BEGIN
  SCRCONTROL(18,0,0);
END;     (* inverse *)

{-----------------------------------------------------------------------------}

(* switch back to normal screen *)
(* white on black *)
PROCEDURE NORMAL;
BEGIN
  SCRCONTROL(17,0,0);
END;
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : TMGR.DIR/T.SUBRT.TEXT          Volume : TFILES             *)
(*      Codefile : T.MGR.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(* File last modified : Feb 23,1983                              NPROC        *)
(******************************************************************************)

(* change the name of a subtest or change screen format or change time cutoff *)
(* for timed subtests.  *)
PROCEDURE CHANGETESTNAME;
VAR TESTSTR : STRING;


    (* give instructions *)
    PROCEDURE GINSTRUCT;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(18,0);
      WRITE('SUBTEST NAME FORMAT INSTRUCTIONS');
      GOTOXY(0,2);
      WRITELN(
  'The first character of the subtest name specifies which screen format the ');
      WRITELN(
      'question text will appear in.  The screen codes are listed below.  The ');
      WRITELN(
  'second character of the subtest name specifies the total permitted time ');
      WRITELN(
  'in minutes for timed subtests.  The second and third characters specify');
      WRITELN(
  'the additional seconds permitted for the timed tests.  Type in the subtest');
      WRITELN(
  'name then press <RET>.');

      WRITELN;
      WRITELN('''*'' as first char = inverse + 80 columns');
      WRITELN('''@'' as first char = normal + 80 columns');
      WRITELN('''?'' as first char = normal + 40 columns');
      WRITELN;
      WRITELN(
 'If first character of test name is not any of the above control characters,');
  WRITELN('then the text format will default to inverse + 40 columns.');
      WRITELN;
      WRITELN;
      WRITELN('Change ',DIRECTORY.TESTNAME,' to what?');
      WRITELN;
      WRITE('-----> ');
    END;  (* ginstruct *)


BEGIN  (* changetestname *)
  (* get the subtest you want to change *)
  LOADTEST('Change name of which subtest? : ');

  IF ESCPROC THEN EXIT(CHANGETESTNAME);

  (* give instructions *)
  GINSTRUCT;

  (* get the new test name *)
  READLN(TESTSTR);

  IF TESTSTR = '' THEN TESTSTR := DIRECTORY.TESTNAME;

  WRITELN;
  WRITELN(DIRECTORY.TESTNAME,' changed to ',TESTSTR);

  (* save the new name in the files *)
  DIRECTORY.TESTNAME := TESTSTR;
  DIRINFO[CURRINDEXRECNUM].TNAME := TESTSTR;
  UPDATEDIRECTORY(CURRINDEXRECNUM);
  WRITELN;
  STALL;
END;  (* change test name *)

{--------------------------------------------------------------------------------}


(**** show main menu command level selections ****)
PROCEDURE MENU;
VAR X,Y : INTEGER;
BEGIN
  PAGE(OUTPUT);
  GOTOXY(20,0);
  WRITE('TEST MANAGER MENU ',VERSION);
  GOTOXY(0,4);
  WRITE('Select one of the following procedures by entering its number.');
  X := 16;
  Y := 8;
  GOTOXY(X,Y);
  WRITE('1.  QUIT');
  GOTOXY(X,Y+1);
  WRITE('2.  FETCH A SUBTEST');
  GOTOXY(X,Y+2);
  WRITE('3.  LIST DIRECTORY');
  COTOXY(X,Y+3);
  WRITE('4.  CREATE NEW SUBTEST');
  GOTOXY(X,Y+4);
  WRITE('5.  CHANGE SUBTEST NAME/FORMAT');
  GOTOXY(X,Y+5);
  WRITE('6.  TRANSFER SUBTEST TO FLOPPY');
  GOTOXY(X,Y+6);
  WRITE('7.  DELETE SUBTEST');
  GOTOXY(X,Y+7);
  WRITE('8.  SEARCH TEXT FOR KEYWORDS');
  GOTOXY(X,Y+10);
  WRITE('Enter Choice # : ');
END;   (* menu *)

{--------------------------------------------------------------------------------}

(* writes the directory information to record   *)
(* puts necessary file info in main memory      *)
PROCEDURE GETDIRINFO;
VAR I,K,ICOUNT : INTEGER;
BEGIN

  (* initialize the directory information *)
  FOR I := 0 TO MAXSUBTESTS DO
    DIRINFO[I].NOTUSED := TRUE;

  (* get the directory information *)
  I := 0;
  RESET(FILEDIRECTORY,INDEXNAME);
  REPEAT
    SEEK(FILEDIRECTORY,I);
    GET(FILEDIRECTORY);
    IF NOT (FILEDIRECTORY^.UNUSED) THEN
      BEGIN (1)
        DIRECTORY := FILEDIRECTORY^;
        DIRINFO[I].NOTUSED := FALSE;
        DIRINFO[I].TNAME := DIRECTORY.TESTNAME;
        ICOUNT := 0;
        FOR K := (MAXSAMPLES+1) TO MAXITEMPOOL DO
          IF DIRECTORY.ITEMCODE[K] > 0
           THEN
               ICOUNT := ICOUNT + 1;
        DIRINFO[I].ITEMCOUNT := ICOUNT;
      END;  (1)
    I := I + 1;
  UNTIL I > MAXSUBTESTS;
  CLOSE(FILEDIRECTORY,NORMAL);
END;  (* getdirinfo *)

{--------------------------------------------------------------------------------}
```

```
(* returns record # of question data file no collisions.  This is a mapping   *)
(* function which takes the location a question code exists in a subtest      *)
(* directory, the maximum questions per subtest and the subtest record number *)
(* and maps it to a location in a file with data for that question           *)
FUNCTION HASH;
BEGIN
  HASH :=
    (CURRINDEXRECNUM * MAXITEMPOOL)
    + KEY + CURRINDEXRECNUM;
END;  (* hash *)

{-------------------------------------------------------------------------------}

(* reads the item text file & displays item text *)
PROCEDURE DECODEPRINT;
CONST MAXBUFSIZE = 2047;
VAR X,
    Y,
    B,
    CURRPTR,
    CURRBLK,
    CHARCODE,
    CHARCNT : INTEGER;

    BADIO : BOOLEAN;

    TEXT[illegible]: STRING[80];


    {---------------------------------------------------------------------------}

    (* return the next code in ascii file *)
    FUNCTION BUFCODE : INTEGER;
    BEGIN
      BUFCODE := TRIX.ASCIIBUF[CURRPTR];
      CURRPTR := CURRPTR + 1;
      IF CURRPTR > 2047 THEN
      (* end of block/get next block and reset byte ptr *)
        BEGIN {1}
          CURRBLK := CURRBLK + 4;
          READITEMBLOCK(CURRBLK);
          CURRPTR := 0;
        END;  {1}
    END;  (* bufcode *)

{...............................................................................}

BEGIN  (* decode print *)
  TEXTSTR :=
'                                                                                ';
  SETSCREEN;  (* clear the screen *)
  READITEMBLOCK(BLOCKNUM);
  (* set block/byte ptrs *)
  CURRPTR := BLOCKPTR;
  CURRBLK := BLOCKNUM;
  FILLCHAR(LINEBUF[0],80,' ');
  (* read bytes from the buffer *)
  REPEAT

    (* get char from block buffer *)
    CHARCODE := bufcode;

    CASE CHARCODE OF
      GOTOFLAG : BEGIN {1}     (* move cursor *)
                   (* next two bytes after flag are x,y coord *)
                   X := BUFCODE;
                   Y := BUFCODE;
                   CHARCNT := BUFCODE;
                   IF (CURRPTR + CHARCNT - 1) > MAXBUFSIZE THEN
                     BEGIN {2}
                       B := (MAXBUFSIZE + 1) - CURRPTR;
                       MOVELEFT(TRIX.ASCIIBUF[CURRPTR],LINEBUF[X],B);
                       X := X + B;
                       B := CHARCNT - B;
                       CURRBLK := CURRBLK + 4;
                       READITEMBLOCK(CURRBLK);
                       CURRPTR := 0;
                       MOVELEFT(TRIX.ASCIIBUF[CURRPTR],LINEBUF[X],B);
                       CURRPTR := CURRPTR + B;
                     END    {2}
                     ELSE
                     BEGIN {3}
                       MOVELEFT(TRIX.ASCIIBUF[CURRPTR],LINEBUF[X],CHARCNT);
                       CURRPTR := CURRPTR + CHARCNT;
                       IF CURRPTR > MAXBUFSIZE THEN
                         BEGIN {4}
                           CURRBLK :=  CURRBLK + 4;
                           CURRPTR := 0;
                           READITEMBLOCK(CURRBLK);
                         END;  {4}
                     END;  {3}

                     GOTOXY(0,Y);
                     WRITE(LINEBUF);
                     FILLCHAR(LINEBUF[0],80,' ');
                 END;  {1}
      PAGEFLAG : BEGIN {5}    (* wait for keystroke to see rest of text *)
                   GOTOXY(1,21);
                   STALL;
                   PAGE(OUTPUT);
                 END;  {5}
      ENDITEM  : ;

    END;
  UNTIL CHARCODE = ENDITEM; (* until end flag hit *)
END;  (* decodeprint *)

{----------------------------------------------------------------------------}

(* gets the a,b, and c  parameters for items *)
PROCEDURE GETPARAM;
VAR VALUE  : REAL;
BEGIN
  PAGE(OUTPUT);
  WRITE('Enter ',PTYPE,' parameter value, then press <RET> : ');
(*$I-*)
  READLN(VALUE);
(*$I+*)
  CASE PTYPE OF
    'A' : AVALUE := VALUE;
    'B' : BVALUE := VALUE;
    'C' : CVALUE := VALUE;
  END;  (* case *)
END;  (* get param *)

{----------------------------------------------------------------------------}

(* returns ptr to 1st free slot in subtest directory to put item *)
(* code.  Returns nil if no room .                               *)
FUNCTION ITEMFREESLOT;
VAR SLOT : INTEGER;
   FOUND : BOOLEAN;
BEGIN
  SLOT := MAXSAMPLES + 1;
  FOUND := FALSE;
  REPEAT
    IF DIRECTORY.ITEMCODE[SLOT] < 0
      THEN
          FOUND := TRUE
      ELSE
          SLOT := SLOT + 1;
  UNTIL (SLOT > MAXITEMPOOL) OR (FOUND);
  IF FOUND
    THEN
```

```
        ITEMFREESLOT := SLOT
    ELSE
        ITEMFREESLOT := NIL;
END;  (* itemfreeslot *)

{-----------------------------------------------------------------------------------}

(* set the graphics flag *)
PROCEDURE SETGFLAG;
VAR SELECT : CHAR;
BEGIN
  PAGE(OUTPUT);
  GOTOXY(20,0);
  WRITE('SET GRAPHICS FLAG MENU');
  GOTOXY(0,4);
  WRITE('Select one of the following options by entering its number.');
  GOTOXY(16,8);
  WRITE('1.  QUIT');
  GOTOXY(16,9);
  WRITE('2.  GRAPHICS FLAG OFF  (no graphic text)');
  GOTOXY(16,10);
  WRITE('3.  NORMAL GRAPHICS FLAG ON');
  GOTOXY(16,11);
  WRITE('4.  COMPRESSED GRAPHICS FLAG ON');
  REPEAT
    GOTOXY(0,6);
    WRITE('You currently have option ');
    IF NOT ITEMINFO.GRAPHICS THEN
      WRITE('2  (no graphic text)         ')
    ELSE
      IF ITEMINFO.DUMMY1 = COMPRESSED THEN
        WRITE('4  (compressed graphic text)')
      ELSE
        WRITE('3  (normal graphic text)    ');
    INVERSE;
    IF ITEMINFO.GRAPHICS THEN
      BEGIN {1}
        IF ITEMINFO.DUMMY1 = COMPRESSED THEN
        BEGIN
          GOTOXY(16,11);
          WRITE('4.  COMPRESSED GRAPHICS FLAG ON');
        END
        ELSE
        BEGIN
          GOTOXY(16,10);
          WRITE('3.  NORMAL GRAPHICS FLAG ON');
        END;
      END   {1}
      ELSE
      BEGIN {2}
        GOTOXY(16,9);
        WRITE('2.  GRAPHICS FLAG OFF  (no graphic text)');
      END;  {2}
    NORMAL;
    GOTOXY(31,14);
    WRITE('    ');
    GOTOXY(16,14);
    WRITE('Enter Choice # : ');
    SELECT := GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
    IF ITEMINFO.GRAPHICS THEN
      BEGIN {3}
        IF ITEMINFO.DUMMY1 = COMPRESSED THEN
        BEGIN
          GOTOXY(16,11);
          WRITE('4.  COMPRESSED GRAPHICS FLAG ON');
        END
        ELSE
        BEGIN
          GOTOXY(16,10);
          WRITE('3.  NORMAL GRAPHICS FLAG ON');
        END;
      END   {3}
      ELSE
      BEGIN {4}
        GOTOXY(16,9);
        WRITE('2.  GRAPHICS FLAG OFF  (no graphic text)');
      END;  {4}
    CASE SELECT OF
      '1' : ;
      '2' : ITEMINFO.GRAPHICS := FALSE;
      '3' : BEGIN
              ITEMINFO.GRAPHICS := TRUE;
              ITEMINFO.DUMMY1 := 0.0;
            END;
      '4' : BEGIN
              ITEMINFO.GRAPHICS := TRUE;
              ITEMINFO.DUMMY1 := COMPRESSED;
            END;
    END;  (* cases *)
  UNTIL SELECT = '1';
END;  (* setgflag *)

{---------------------------------------------------------------------------}

(* this procedure changes the screen based on global vars *)
PROCEDURE SETSCREEN;
BEGIN

  (* set the column length *)
  IF SCR80 THEN
    BEGIN {1}
      TEXT80MODE;
      RIGHTMAX := 78;
    END   {1}
    ELSE
    BEGIN {2}
      TEXT40MODE;
      RIGHTMAX := 38;
    END;  {2}

  (* set the video display *)
  IF VNORMAL
    THEN
      NORMAL
    ELSE
      INVERSE;

  PAGE(OUTPUT);
END;    (* setscreen *)

{---------------------------------------------------------------------------}

(* does a write to graphics screen for string values *)
PROCEDURE GWRITESTR;
BEGIN
  UNITWRITE(3,GSTR[1],LENGTH(GSTR),0,12);
END;  (* gwritestr *)

{---------------------------------------------------------------------------}

(* do a gotoxy to the graphics screen, treats graphics screen as if it *)
(* had textmode coordinates for 80 column by 24 rows.                  *)
PROCEDURE GGOTOXY;
VAR XPOS,
    YPOS : INTEGER;
BEGIN
  XPOS := X * 7;
  YPOS := 191 - (Y * 8);
  MOVETO(XPOS,YPOS)
END;  (* ggotoxy *)

{---------------------------------------------------------------------------}

(* set up the grafix mode *)
PROCEDURE INITFORGRAFIX;
BEGIN
```

```
  INITGRAFIX;
  GRAFIXMODE(BW560,1);
  VIEWPORT(0,559,0,191);
  FILLCOLOR(WHITE);
  PENCOLOR(BLACK);
  FILLPORT;
END;  (* initforgrafix *)

 {----------------------------------------------------------------------------}

(* does a write to graphics screen for char values *)
PROCEDURE GWRITECHR;
VAR C : STRING;
BEGIN
  C := ' ';
  C[1] := GCHR;
  UNITWRITE(3,C[1],1,0,12);
END;  (* gwritechr *)

 {----------------------------------------------------------------------------}

(* does a write to graphics screen for integer values *)
PROCEDURE GWRITEINT;
VAR X,Z : INTEGER;
    DIGITSTR,STR : STRING;
    NEGATIVE : BOOLEAN;
    C : CHAR;
BEGIN
  NEGATIVE := FALSE;
  Z := GINT;
  IF GINT < 0 THEN
  BEGIN
    NEGATIVE := TRUE;
    Z := -GINT;
  END;
  DIGITSTR := ' ';
  STR := '';
  REPEAT
    X := Z MOD 10;
    C := CHR(X+48);
    DIGITSTR[1] := C;
    STR := CONCAT(DIGITSTR,STR);
    Z := Z DIV 10;
  UNTIL Z <= 0;
  IF NEGATIVE THEN
    STR := CONCAT('-',STR);
  UNITWRITE(3,STR[1],LENGTH(STR),0,12);
END;   (* gwriteint *)

 {----------------------------------------------------------------------------}

(* do  a writeln to the graphics screen *)
PROCEDURE GWRITELN;
BEGIN
  MOVETO(0,YLOC-8);
END;  (* gwriteln *)

 {----------------------------------------------------------------------------}

(* does a stall to the graphics screen *)
PROCEDURE GSTALL;
BEGIN
  GWRITESTR('Press <RET> to continue ');
  IF GETCHAR([CHR(RET)],TRUE,FALSE,TRUE) = CHR(RET) THEN;
END;     (* gstall *)

 {----------------------------------------------------------------------------}

(* blanklines on the graphics screen, treat as if in textmode  *)
PROCEDURE GBLANKLINES;
VAR TOP,
    BOTTOM : INTEGER;
BEGIN
```

```
  TOP := 191 - (STARTBLANK * 8);
  BOTTOM := 191 - ((STARTBLANK + BLANKTHISMANY) * 8);
  VIEWPORT(0,559,BOTTOM,TOP);
  FILLPORT;
  VIEWPORT(0,559,0,191);
  GGOTOXY(0,LEAVECURSOR);
END;   (* gblanklines *)

{-----------------------------------------------------------------------------}


(* display question to graphics screen *)
PROCEDURE GDECODEPRINT;
VAR X,Z : INTEGER;
    VNAME,FNAME,DIGITSTR,STR : STRING;
    C : CHAR;


    (* reads the item text file and displays the graphics *)
    PROCEDURE DECODEGRAF;
    VAR X,
        Y,
        X1,Y1,
        CURRPTR,
        CURRBLK,
        DOTCNT : INTEGER;

        (* reads a block from disk into the item ascii buffer *)
        PROCEDURE READITEMBLOCK(WHICHBLOCK : INTEGER);
        VAR BLOCKSTRANSFERRED : INTEGER;
            BADIO : BOOLEAN;
        BEGIN
          BADIO := FALSE;
          RESET(ITEMTEXT,FNAME);
          BLOCKSTRANSFERRED := BLOCKREAD(ITEMTEXT,TRIX.ITEMBUF,4,WHICHBLOCK);
          BADIO := ((BLOCKSTRANSFERRED < 4) OR (IORESULT <> 0));
          CLOSE(ITEMTEXT,LOCK);
          IF BADIO THEN
          BEGIN
            WRITELN;
            WRITELN;
            WRITE(' Block ', WHICHBLOCK,' write error.');
            WRITELN;
            READLN;
            EXIT(PROGRAM);
          END;
        END;  (* readitemblock  *)


        (* reads the item text file & displays item text *)
        PROCEDURE DECODEPRINT;
        VAR X,
            Y,
            BYTECNT,
            CHARCODE : INTEGER;


            (* return the next code in ascii file *)
            FUNCTION BUFCODE : INTEGER;
            BEGIN
              BUFCODE := TRIX.ASCIIBUF[CURRPTR];
              CURRPTR := CURRPTR + 1;
              IF CURRPTR > 2047 THEN
              (* end of block/get next block and reset byte ptr *)
                BEGIN {1}
                  CURRBLK := CURRBLK + 4;
                  READITEMBLOCK(CURRBLK);
                  CURRPTR := 0;
                END; {1}
            END;  (* bufcode *)
```

```
      BEGIN  (* decode print *)
        (* read bytes from the buffer *)
        REPEAT

          (* get char from block buffer *)
          CHARCODE := bufcode;

          CASE CHARCODE OF
            GOTOFLAG : BEGIN (1)     (* move cursor *)
                         (* next two bytes after flag are x,y coord *)
                         X := BUFCODE;
                         Y := BUFCODE;
                         BYTECNT := BUFCODE;
                         GGOTOXY(X,Y);
                       END;  (1)
            ENDITEM  : ;

          END;

          IF (CHARCODE >= 32) AND (CHARCODE <= 126) THEN
            GWRITECHR(CHR(CHARCODE));

        UNTIL CHARCODE = ENDITEM; (* until end flag hit *)
      END;  (* decodeprint *)



    BEGIN  (* decode graf *)

      READITEMBLOCK(0);

      CURRBLK := 0;
      CURRPTR := 0;


      (* decode the xlines *)
      REPEAT
        IF CURRPTR > 1021 THEN
        (* end of block/get new block *)
        BEGIN
          CURRBLK := CURRBLK + 4;
          READITEMBLOCK(CURRBLK);
          CURRPTR := 0;
        END;
        X := TRIX.ITEMBUF[CURRPTR];
        IF X >= 0 THEN
        BEGIN
          Y := TRIX.ITEMBUF[CURRPTR + 1];
          MOVETO(X,Y);
          DOTCNT := TRIX.ITEMBUF[CURRPTR + 2];
          LINEREL(DOTCNT,0);
          CURRPTR := CURRPTR + 3;
        END;
      UNTIL X < 0;

      CURRPTR := CURRPTR + 1;

      (* decode the ylines *)
      REPEAT
        IF CURRPTR > 1021 THEN
        (* end of block/get new block *)
        BEGIN
          CURRBLK := CURRBLK + 4;
          READITEMBLOCK(CURRBLK);
          CURRPTR := 0;
        END;
        X := TRIX.ITEMBUF[CURRPTR];
        IF X >= 0 THEN
        BEGIN
          Y := TRIX.ITEMBUF[CURRPTR + 1];
          MOVETO(X,Y);
```

```
           DOTCNT := TRIX.ITEMBUF[CURRPTR + 2];
           LINEREL(0,DOTCNT);
           CURRPTR := CURRPTR + 3;
         END;
       UNTIL X < 0;

       CURRPTR := CURRPTR + 1;


       (* decode the diagonals *)
       REPEAT
         IF CURRPTR > 1020 THEN
         (* end of block/get new block *)
         BEGIN
           CURRBLK := CURRBLK + 4;
           READITEMBLOCK(CURRBLK);
           CURRPTR := 0;
         END;
         X := TRIX.ITEMBUF[CURRPTR];
         IF X >= 0 THEN
         BEGIN
           Y := TRIX.ITEMBUF[CURRPTR + 1];
           X1 := TRIX.ITEMBUF[CURRPTR + 2];
           Y1 := TRIX.ITEMBUF[CURRPTR + 3];
           MOVETO(X,Y);
           LINETO(X1,Y1);
           CURRPTR := CURRPTR + 4;
         END;
       UNTIL X < 0;

       CURRPTR := CURRPTR + 1;

       (* decode the dots *)
       REPEAT
         IF CURRPTR > 1022 THEN
         (* end of block/get new block *)
         BEGIN
           CURRBLK := CURRBLK + 4;
           READITEMBLOCK(CURRBLK);
           CURRPTR := 0;
         END;
         X := TRIX.ITEMBUF[CURRPTR];
         IF X >= 0 THEN
         BEGIN
           Y := TRIX.ITEMBUF[CURRPTR + 1];
           DOTAT(X,Y);
           CURRPTR := CURRPTR + 2;
         END;
       UNTIL X < 0;

       CURRPTR := CURRPTR + 1;
       CURRPTR := CURRPTR * 2;

       DECODEPRINT;

     END;   (* decodegraf *)


BEGIN
  FILLPORT;
  GRAFIXON;
  DIGITSTR := ' ';
  STR := '';
  Z := ITEMCODE;
  REPEAT
    X := Z MOD 10;
    C := CHR(X+48);
    DIGITSTR[1] := C;
    STR := CONCAT(DIGITSTR,STR);
    Z := Z DIV 10;
  UNTIL Z <= 0;
  DIGITSTR[1] := CHR(SUBTESTNUM+65);
  C := DIGITSTR[1];
  (*
  IF SAMPLEQUESTION THEN
  BEGIN
    FNAME := CONCAT('/CATFOTO/',DIGITSTR,'DIR/G',DIGITSTR,'SD',STR,'.FOTO');
    GLOAD(FNAME);
  END
  ELSE
  *)

  IF ITEMINFO.DUMMY1 = COMPRESSED THEN
  BEGIN
    FNAME := CONCAT('/CATFOTO/',DIGITSTR,'DIR/G',DIGITSTR,STR,'.DATA');
    DECODEGRAF;
  END
  ELSE
  BEGIN
    FNAME := CONCAT('/CATFOTO/',DIGITSTR,'DIR/G',DIGITSTR,'Q',STR,'.FOTO');
    GLOAD(FNAME);
  END;
END;    (* gdecodeprint *)
```

```
(*************************************************************************)
(*                                                                       *)
(*     Textfile : TMGR.DIR/T.1SUBRT.TEXT         Volume : TFILES         *)
(*     Codefile : T.MGR.CODE ('Include' file)    Volume : CATDATA        *)
(*                                                                       *)
(*************************************************************************)
(* File last modified : MAY 25, 1983             NPRDC                   *)
(*************************************************************************)


(* codes the screen array into compact block buffer and writes it to disk.  *)
SEGMENT PROCEDURE CODESCREEN(ENDOFITEM : BOOLEAN);
VAR X0,
    CHARCODE,X,Y,I,
    BLOCK,
    BYTE,
    BYTECOUNT : INTEGER;
DONE : BOOLEAN;

    {.....................................................................}

    (* fill buffer with ascii & codes, write to disk  *)
    PROCEDURE FILLITEMBUFFER(BUFFCODE : INTEGER);
    BEGIN
      TRIX.ASCIIBUF[CURRFREEPTR] := BUFFCODE;
      CURRFREEPTR := CURRFREEPTR + 1;
      IF CURRFREEPTR >= 2048 THEN
        BEGIN {1}
          WRITEITEMBLOCK(CURRBLOCK);
          CURRBLOCK := CURRBLOCK + 4;
          CURRFREEPTR := 0;
        END;  {1}
    END;  (* fillitembuffer *)

{.........................................................................}

BEGIN  (* codescreen *)

 (* read in current block where text will start,  into buffer *)
 READITEMBLOCK(CURRBLOCK);

 (* display message *)
 PAGE(OUTPUT);
 GOTOXY(1,1);
 WRITE('Entering text');

 (* start at line 0 of screen buffer *)
 Y := 0;
 REPEAT

   (* point to first character on line *)
   X := 0;
   WRITE('.');

   (* look for leading blanks *)
   DONE := FALSE;
   REPEAT
     IF SCREEN[X,Y] = CHR(SPACE)
       THEN
           X := X + 1
       ELSE
           DONE := TRUE;
   UNTIL  (X > XSCREEN) OR (DONE);

   (* if the whole line was not blank  *)
   IF X <= XSCREEN THEN
   BEGIN {1}

     (* flag a gotoxy where first non-blank character begins *)
     FILLITEMBUFFER(GOTOFLAG);
     FILLITEMBUFFER(X);
     FILLITEMBUFFER(Y);
     BYTECOUNT := 0;
    (* figure out how many bytes on this line *)
    X0 := X;

    REPEAT
      I := X0;
      DONE := FALSE;

      REPEAT
        IF SCREEN[I,Y] = CHR(SPACE)
          THEN
              I := I + 1
          ELSE
              DONE := TRUE;
      UNTIL  (I > XSCREEN) OR (DONE);

      IF I < XSCREEN THEN
        BEGIN {2}
          BYTECOUNT := BYTECOUNT + 1;
          X0 := X0 + 1;
          DONE := FALSE;
        END   {2}
        ELSE
            DONE := TRUE;
    UNTIL (DONE) OR (X0 > XSCREEN);

    FILLITEMBUFFER(BYTECOUNT);

    REPEAT

      (* look for trailing blanks *)
      I := X;
      DONE := FALSE;

      REPEAT
        IF SCREEN[I,Y] = CHR(SPACE)
          THEN
              I := I + 1
          ELSE
              DONE := TRUE;
      UNTIL  (I > XSCREEN) OR (DONE);

      (* if the rest of the line was not all blanks *)
      IF I < XSCREEN THEN
        BEGIN {3}

          (* save the character in the block buffer *)
          FILLITEMBUFFER(ORD(SCREEN[X,Y]));

          (* set pointer to next character *)
          X := X + 1;

          (* set loop condition to process next character *)
          DONE := FALSE;
        END   {3}
        ELSE

            (* done with this line, go look at next line in screen buffer *)
            DONE := TRUE;

    (* until done with line or last character in line processed *)
    UNTIL (DONE) OR (X > XSCREEN);
  END;  {1}

  (* point to next line *)
  Y := Y + 1;

(* until all lines looked at *)
UNTIL Y >= YSCREEN;

(* if editing was cancelled with a control - c *)
IF ENDOFITEM
  THEN
```

```
       (* flag end of question text *)
       FILLITEMBUFFER(ENDITEM)
   ELSE
       (* editing was temporarily cancelled with a control - p *)
       (* flag that text continues on another page.             *)
       FILLITEMBUFFER(PAGEFLAG);

  (* write the last block of ascii to file *)
 WRITEITEMBLOCK(CURRBLOCK);

END;  (* codescreen *)

{----------------------------------------------------------------------------}

(* fills an array representing the crt with char screen location may be       *)
(* specified this is the guts of the question text editor  *)
SEGMENT PROCEDURE FILLSCREENBUFFER(UPBOUND,RIGHTBOUND,
                                   LOBOUND,LEFTBOUND : INTEGER;
                                   FLUSHBUF : BOOLEAN);
VAR SCREENCHAR : CHAR;
    CONTROLCHAR : SETOFCHAR;
    SCREENBYTES,
    CHARCODE,
    X,
    Y,
    L,
    RIGHT : INTEGER;
    MORE : BOOLEAN;

BEGIN

  (* save right boundary *)
  RIGHT := RIGHTBOUND;

  (* initialize indicator for additional lines *)
  MORE := FALSE;

  (* initialize the page count *)
  PAGENUM := 1;

  (* set # of bytes in screen character buffer *)
  SCREENBYTES := (XSCREEN + 1) * (YSCREEN + 1);


  (* if wish to start with a blank buffer, new text *)
  IF FLUSHBUF THEN
    BEGIN {1}
      (* clear screen *)
      FILLCHAR(SCREEN[0],SCREENBYTES,' ');

      (* set the screen parameters *)
      SETSCREEN;
    END;  {1}

  (* put cursor in upper left hand corner *)
  X := LEFTBOUND;
  Y := UPBOUND;

  CONTROLCHAR := [CHR(UP),CHR(DOWN),CHR(LARROW),CHR(RARROW),
                  CHR(ETX),CHR(PAGEOUT),CHR(RET),CHR(ESC)];

  CHARCODE := PAGEOUT;

  BLANKLINES(20,4,21);

  (* fill up the screen character buffer until done *)
  REPEAT

    (* page number and instructions  *)
    IF CHARCODE = PAGEOUT THEN
      BEGIN {2}
        IF MORE THEN GOTOXY(0,2)
        ELSE GOTOXY(0,20);
    FOR L := 1 TO RIGHT DO
      WRITE('-');WRITELN('-');
    WRITELN(' Enter text. Use arrows to move cursor.');
    WRITELN(' <RET>:next line, <CTRL>-P:new page.');
    WRITE(' <CTRL>-C to quit and save.  ');
  END;  (2)

(* monitor cursor location *)
GOTOXY(X,Y);

(* get a character from the keyboard *)
SCREENCHAR := GETCHAR(CHARACTERS + CONTROLCHAR,TRUE,TRUE,TRUE);


(* get the ascii value *)
CHARCODE := ORD(SCREENCHAR);

(* check for cursor control characters *)
CASE CHARCODE OF

  (* cursor moved up but not beyond set boundaries *)
  UP : IF Y <= UPBOUND
         THEN
             SQUAWK
         ELSE
             Y := Y - 1;

  (* cursor moved down but not beyond set boundaries *)
  DOWN : IF Y >= LOBOUND
           THEN
               SQUAWK
           ELSE
               IF MORE THEN
                 BEGIN
                   IF Y >= 1   (* gives 2 lines at top *)
                     THEN SQUAWK
                     ELSE
                     Y := Y + 1
                 END
                 ELSE
                 Y := Y + 1;


  (* cursor moved to left with auto wrap around *)
  LARROW : IF X <= LEFTBOUND THEN
           BEGIN (3)
             IF Y <= UPBOUND
               THEN
                   SQUAWK
               ELSE
               BEGIN (4)
                 X := RIGHT;
                 Y := Y - 1;
               END;  (4)
           END   (3)
           ELSE
               X := X - 1;

  (* cursor moved to right with auto wraparound *)
  RARROW : IF X >= RIGHT THEN
             BEGIN (5)
               IF (Y >= LOBOUND) OR (MORE AND (Y >=1))
                 THEN
                     SQUAWK
                 ELSE
                 BEGIN (6)
                   Y := Y + 1;
                   X := LEFTBOUND;
                 END;  (6)
             END   (5)
             ELSE
               X := X + 1;
    (* carriage return *)     (* if at bottom line then make room *)
    RET : IF (Y >= LOBOUND) OR MORE
           THEN
            BEGIN
              IF  NOT PCRECNUM
              THEN
                  SQUAWK
              ELSE
               IF MORE AND (Y >= 1)  (* border after 2 lines *)
                 THEN
                     SQUAWK
                 ELSE
                  IF NOT MORE THEN
                     BEGIN
                       MORE := TRUE;
                       CHARCODE := PAGEOUT;
                       PAGE(OUTPUT);
                       IF NOT FLUSHBUF THEN
                         BEGIN
                           GOTOXY(LEFTBOUND,UPBOUND);
                           FOR X:= LEFTBOUND TO RIGHT DO WRITE(SCREEN[X,20]);
                           WRITELN;
                           FOR X:= LEFTBOUND TO RIGHT DO WRITE(SCREEN[X,21]);
                         END;
                       X := LEFTBOUND;
                       Y := UPBOUND;
                     END
                   ELSE
                     BEGIN
                          Y := Y + 1;
                          X := LEFTBOUND;
                     END
              END

             ELSE
             BEGIN {7}
               Y := Y + 1;
               X := LEFTBOUND;
             END;  {7}

      (* start new page *)
      PAGEOUT : BEGIN {8}
                  BLANKLINES(21,3,21);

                  (* check if satisfied with current page *)
                  WRITE(' Page ',PAGENUM,' Okay?  Y/N : ');
                  IF GETCHAR(['y','n','Y','N'],
                             TRUE,TRUE,TRUE) IN ['y','Y'] THEN
                    BEGIN {9}

                      (* write current page out to disk *)
                      CODESCREEN(FALSE);

                      (* clear the screen and begin new page editing *)
                      PAGE(OUTPUT);
                      PAGENUM := PAGENUM + 1;

                      (* reinitialize the screen buffer to all blanks *)
                      FILLCHAR(SCREEN[0],SCREENBYTES,' ');
                      X := LEFTBOUND;
                      Y := UPBOUND;
                    END;  {9}
                END;  {8}
    END;  (* cases *)

    (* if character typed was a visible character *)
    IF (CHARCODE >= 32) AND (CHARCODE <= 126) THEN
    BEGIN {10}

      (* if last character exceeded screen boundaries *)
      IF (X >= RIGHT) AND ((Y >= LOBOUND) OR (MORE AND (Y >=1)))
        THEN
            SQUAWK
    ELSE
    BEGIN {11}

     (* save the character in the screen buffer *)
     IF MORE THEN SCREEN[X,Y+20] := SCREENCHAR
     ELSE
     SCREEN[X,Y] := SCREENCHAR;


     (* move cursor over one *)
     X := X + 1;

     (* check auto wrap around *)
     IF X > RIGHT THEN
      BEGIN {12}
       IF Y < LOBOUND THEN
        BEGIN {13}
         X := LEFTBOUND;
         Y := Y + 1;
        END   {13}
        ELSE
          X := X - 1;
      END;  {12}
    END;  {11}
  END;  {10}

 (* until control-c pressed *)
 UNTIL (CHARCODE = ETX) ;


END;  (* fillscreenbuffer *)

{----------------------------------------------------------------------------}

(* select the answer type *)
SEGMENT PROCEDURE SELECTATYPE;
VAR LBOUND,
  HBOUND,
  SELECT : CHAR;
  ERR : BOOLEAN;
  I : INTEGER;

  {.......................................................................}

  (* get the answer bounds *)
  PROCEDURE GETABOUND;
  BEGIN
   ERR := FALSE;
   REPEAT
    PAGE(OUTPUT);
    GOTOXY(22,0);
    WRITE('ENTER ANSWER SELECTION RANGE');
    GOTOXY(0,3);
    WRITE('Low bound?  (letter/digit) : ');
    LBOUND := GETCHAR(['0'..'9','A'..'Z'],TRUE,TRUE,TRUE);
    WRITELN;
    WRITE('High bound?  (letter/digit) : ');
    HBOUND := GETCHAR(['0'..'9','A'..'Z'],TRUE,TRUE,TRUE);
    ERR := FALSE;
    IF (HBOUND IN ['0'..'9']) THEN {a}
     IF NOT (LBOUND IN ['0'..'9']) THEN {b}
      ERR := TRUE;
    IF (HBOUND IN ['A'..'Z']) THEN {c}
     IF NOT (LBOUND IN ['A'..'Z']) THEN {d}
      ERR := TRUE;
    IF ERR THEN {e}
     BEGIN {1}
      WRITELN;
      WRITELN;
      WRITELN('Bounds range type mismatch error!');
      SQUAWK;
      WRITELN;
      STALL;
```

164

```
          END;  {1}

        IF (ORD(LBOUND) > ORD(HBOUND)) AND (NOT ERR) THEN {f}
          BEGIN {2}
            WRITELN;
            WRITELN;
            WRITELN('Low bound exceeds high bound error!');
            SQUAWK;
            WRITELN;
            STALL;
            ERR := TRUE;
          END;  {2}
      UNTIL NOT ERR;
      ITEMINFO.HIGHANSWER := HBOUND;
      ITEMINFO.LOWANSWER := LBOUND;
    END;   (* getabound *)

{.............................................................................}

BEGIN  (* selectatype *)
  PAGE(OUTPUT);
  GOTOXY(20,0);
  WRITE('SELECT ANSWER TYPE');
  GOTOXY(0,4);
  WRITE('Select one of the following options by entering its number.');
  GOTOXY(16,8);
  WRITE('1.  MULTIPLE CHOICE/SINGLE ANSWER');
  GOTOXY(16,9);
  WRITE('2.  INTEGER VALUE ANSWER');
  GOTOXY(16,10);
  WRITE('3.  MULTIPLE CHOICE/MULTIPLE ANSWERS');
  GOTOXY(16,14);
  WRITE('Enter Choice # : ');
  SELECT := GETCHAR(['1'..'3'],TRUE,TRUE,TRUE);
  IF SELECT <> '2' THEN GETABOUND;

  PAGE(OUTPUT);
  CASE SELECT OF
   '1' : BEGIN {1}
           GOTOXY(20,0);
           WRITE('ENTER MULTIPLE CHOICE/SINGLE ANSWER');
           GOTOXY(0,3);
           WRITELN('The answer range is : ',ITEMINFO.LOWANSWER,'..',
                    ITEMINFO.HIGHANSWER);
           WRITELN;
           ITEMINFO.ATYPE := CHARVALUE;
           GOTOXY(0,6);
           WRITE('Enter answer and then press <RET> : ');
           FILLBUF(1,[ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER],
                 TRUE);
           ITEMINFO.ANSWER := LINEBUF[0];
           LINEBUF[0] := ' ';
         END;  {1}
   '2' : BEGIN {2}
           GOTOXY(20,0);
           WRITE('ENTER INTEGER VALUE ANSWER');
           ITEMINFO.ATYPE := INTVALUE;
           GOTOXY(0,6);
           WRITE('Enter answer (integer ',
                 'value) and then press <RET> : ');
           READLN(I);
           ITEMINFO.INTANSWER := I;
         END;  {2}
   '3' : BEGIN {3}
           GOTOXY(20,0);
           WRITE('ENTER MULTIPLE CHOICE/MULTIPLE ANSWERS');
           GOTOXY(0,3);
           WRITE('The answer range is : ',ITEMINFO.LOWANSWER,'..',
                    ITEMINFO.HIGHANSWER);
           ITEMINFO.ATYPE := SEVENCHR;
           GOTOXY(0,6);
           WRITE('Enter the number of answers',
                 ' this question has (1..7) : ');
          SELECT := GETCHAR(['1'..'7'],TRUE,TRUE,TRUE);
          ITEMINFO.ANSWERCOUNT := ORD(SELECT) - 48;
          WRITELN;
          WRITELN;
          FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
          BEGIN {4}
            WRITE('Enter answer ',I,', then press <RET> : ');
            FILLBUF(1,
                    [ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER],
                    TRUE);
            ITEMINFO.CHRANSWER[I] := LINEBUF[0];
            LINEBUF[0] := ' ';
            WRITELN;
          END;  {4}
        END;  {3}
  END;  (* case *)

END;  (* selectatype  *)

{--------------------------------------------------------------------------}

(* gets question data  and text and puts into file *)
SEGMENT PROCEDURE INSERTITEM;
VAR OPENSLOT,
    HASHLOC,
    CODENUM,
    SLOT  : INTEGER;
    DONE,
    CODEOK : BOOLEAN;

    {..................................................................}

    (* get data for items *)
    PROCEDURE GETITEMDATA;
    BEGIN  (* get item data *)
      SELECTATYPE;
      (* get item parameters *)
      PAGE(OUTPUT);
      GETPARAM('A');
      GETPARAM('B');
      GETPARAM('C');
      WITH ITEMINFO DO
      BEGIN {1}
        A := AVALUE;
        B := BVALUE;
        C := CVALUE;
      END;  {1}
    END; (* get item data *)

    {------------------------------------------------------------------}

    (* verify the data *)
    PROCEDURE VERIFYDATA;
    VAR SELECT : CHAR;
    BEGIN
      REPEAT
        PAGE(OUTPUT);
        GOTOXY(20,0);
        WRITE('VERIFY QUESTION DATA MENU');
        GOTOXY(0,4);
        WRITE('Select one of the following options by entering its number.');
        GOTOXY(0,6);
        WRITE('Current data : ');
        GOTOXY(15,6);
        WRITE('A parameter = ',ITEMINFO.A:5:3);
        GOTOXY(15,7);
        WRITE('B parameter = ',ITEMINFO.B:5:3);
        GOTOXY(15,8);
        WRITE('C parameter = ',ITEMINFO.C:5:3);
        GOTOXY(16,12);
        WRITE('1.  QUIT  (data ok)');
        GOTOXY(16,13);
        WRITE('2.  CHANGE PARAMETER A');
        GOTOXY(16,14);
        WRITE('3.  CHANGE PARAMETER B');
        GOTOXY(16,15);
        WRITE('4.  CHANGE PARAMETER C');
        GOTOXY(16,18);
        WRITE('Enter Choice # : ');
        SELECT := GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
        CASE SELECT OF
          '1' : ;
          '2' : BEGIN {1}
                  GETPARAM('A');
                  ITEMINFO.A := AVALUE;
                END;  {1}
          '3' : BEGIN {2}
                  GETPARAM('B');
                  ITEMINFO.B := BVALUE;
                END;  {2}
          '4' : BEGIN {3}
                  GETPARAM('C');
                  ITEMINFO.C := CVALUE;
                END;  {3}
        END;  (* cases *)
      UNTIL SELECT = '1';
    END;  (* verify data *)

{............................................................................}

BEGIN  (* insertitem *)

  (* look for free slot in directory *)
  OPENSLOT := ITEMFREESLOT;

  (* if no room in subtest directory *)
  IF OPENSLOT < 0 THEN
    BEGIN {1}
      PAGE(OUTPUT);
      WRITELN;
      WRITELN('No room to add question !! ');
      WRITELN;
      STALL;
      EXIT(INSERTITEM);
    END;  {1}

  (* get question code # *)
  PAGE(OUTPUT);
  CODEOK := FALSE;
  WRITELN;
  REPEAT
    WRITE('Enter the question code #, then press <RET> : ');
  (*$I-*)
    READLN(CODENUM);
  (*$I+*)

    (* if valid code #, >= 0 *)
    IF CODENUM >= 0 THEN
      BEGIN {2}

        (* check if code # previously used in this subtest  *)
        SLOT := SLOTSEARCH(CODENUM);
        IF SLOT <> NIL THEN
          BEGIN {3}
            WRITELN;
            WRITELN('Code # ',CODENUM,' previously used !');
            WRITELN;
            SQUAWK;
          END   {3}
          ELSE
              CODEOK := TRUE;
      END   {2}
      ELSE
          EXIT(INSERTITEM);
  UNTIL CODEOK;
  (* save code number in subtest directory *)
  DIRECTORY.ITEMCODE[OPENSLOT] := CODENUM;

  (* get ptrs to free space to put question text *)
  LOADPTRS;

  (* save these ptrs for this question *)
  ITEMINFO.ITEMPTR := CURRFREEPTR;
  ITEMINFO.ITEMBLOCK := CURRBLOCK;

  (* enter in text for the question *)
  FILLSCREENBUFFER(TOPMAX,RIGHTMAX,
                   BOTTOMMAX,LEFTMAX,TRUE);

  (* set graphics flag *)
  ITEMINFO.GRAPHICS := FALSE;

  TEXT88MODE;

  (* set the graphics flag *)
  SETGFLAG;

  (* get the data for the question *)
  GETITEMDATA;

  (* verify the data *)
  VERIFYDATA;

  (* write the screen buffer to block buffer to ascii file *)
  CODESCREEN(TRUE);

  (* update the new location of free space to put text *)
  SAVEPTRS;

  (* update the subtest directory to disk *)
  UPDATEDIRECTORY(CURRINDEXRECNUM);

  (* update count of items for this subtest *)
  DIRINFO[CURRINDEXRECNUM].ITEMCOUNT :=
      DIRINFO[CURRINDEXRECNUM].ITEMCOUNT + 1;

  (* find the location to put the question data *)
  HASHLOC := HASH(OPENSLOT);

  (* write the record to disk *)
  UPDATEITEMFILE(HASHLOC);
END;  (* insert item *)
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : TMGR.DIR/T.IO.TEXT             Volume : TFILES             *)
(*      Codefile : T.MGR.CODE ('Include' file)    Volume : CATDATA            *)
(*                                                                            *)
(******************************************************************************)
(*  File last modified : Feb 10, 1983            NPROC                        *)
(******************************************************************************)


(* writes the item ascii buffer to diskfile  *)
PROCEDURE WRITEITEMBLOCK;
VAR BLOCKSTRANSFERRED : INTEGER;
    BADIO : BOOLEAN;
BEGIN
  BADIO := FALSE;
  RESET(ITEMTEXT,TEXTNAME);
  BLOCKSTRANSFERRED := BLOCKWRITE(ITEMTEXT,TRIX.ASCIIBUF,4,WHICHBLOCK);
  BADIO := ((BLOCKSTRANSFERRED < 4) OR (IORESULT <> 0));
  CLOSE(ITEMTEXT,LOCK);
  IF BADIO THEN
  BEGIN {1}
    WRITELN;WRITELN;
    WRITE(' Block ',WHICHBLOCK,' write io error.');
    WRITELN;
    WRITELN(' Possibly no room to expand ',TEXTNAME);
    WRITELN(' Must have unused space at end of file');
    WRITELN(' or put file at end of directory.');
    WRITELN;
    STALL;
    EXIT(PROGRAM);
  END;  {1}
END;  (* writeitemblock *)

{------------------------------------------------------------------------------}

(* reads a block from disk into the item ascii buffer *)
PROCEDURE READITEMBLOCK;
VAR BLOCKSTRANSFERRED : INTEGER;
    BADIO : BOOLEAN;
BEGIN
  BADIO := FALSE;
  RESET(ITEMTEXT,TEXTNAME);
  BLOCKSTRANSFERRED := BLOCKREAD(ITEMTEXT,TRIX.ASCIIBUF,4,WHICHBLOCK);
  BADIO := ((BLOCKSTRANSFERRED < 4) OR (IORESULT <> 0));
  CLOSE(ITEMTEXT,LOCK);
  IF BADIO THEN
  BEGIN {1}
    WRITELN;WRITELN;
    WRITE(' Block ',WHICHBLOCK,' read io error.');
    WRITELN;
    WRITELN(' Cant read ',TEXTNAME);
    WRITELN;
    STALL;
    EXIT(PROGRAM);
  END; {1}
END;

{------------------------------------------------------------------------------}

(* returns the slot # in subtest directory where question is, nil if the     *)
(* question code is not in the directory           *)
FUNCTION SLOTSEARCH;
VAR SLOT : INTEGER;
    FOUND : BOOLEAN;
BEGIN

  (* skip space reserved for samples *)
  SLOT := MAXSAMPLES + 1;

  FOUND := FALSE;
  REPEAT
    IF DIRECTORY.ITEMCODE[SLOT] = CODE
```

```
      THEN
          FOUND := TRUE
      ELSE
          SLOT := SLOT + 1;
  UNTIL (SLOT > MAXITEMPOOL) OR (FOUND);
  IF FOUND
    THEN
        SLOTSEARCH := SLOT
    ELSE
        SLOTSEARCH := NIL;
END;  (* slot search *)

{---------------------------------------------------------------------------}

(* updates test directory *)
PROCEDURE UPDATEDIRECTORY;
BEGIN
  RESET(FILEDIRECTORY,INDEXNAME);
  SEEK(FILEDIRECTORY,RECNUM);
  FILEDIRECTORY^ := DIRECTORY;
  PUT(FILEDIRECTORY);
  CLOSE(FILEDIRECTORY,LOCK);
END;  (* updatedirectory *)

{---------------------------------------------------------------------------}

(* updates item data file *)
PROCEDURE UPDATEITEMFILE;
BEGIN
  RESET(FILEITEMINFO,DATANAME);
  SEEK(FILEITEMINFO,RECNUM);
  FILEITEMINFO^ := ITEMINFO;
  PUT(FILEITEMINFO);
  CLOSE(FILEITEMINFO,LOCK);
END;  (* updateitemfile *)

{---------------------------------------------------------------------------}

(* saves value of free space, block & byte ptr in block 0, bytes 0..3 of text *)
(* file expects the block value to be in CURRBLOCK *)
(* and the byte value to be in CURRFREEPTR.   *)
PROCEDURE SAVEPTRS;
VAR TRIX1 : RECORD CASE INTEGER OF
             1 : (TWOBYTES : PACKED ARRAY
                             [0..1] OF CHAR);
             2 : (INTVALUE : INTEGER);
           END;
BEGIN
  READITEMBLOCK(0);
  TRIX1.INTVALUE := CURRBLOCK;
  MOVELEFT(TRIX1.TWOBYTES[0],TRIX.ASCIIBUF[0],2);
  TRIX1.INTVALUE := CURRFREEPTR;
  MOVELEFT(TRIX1.TWOBYTES[0],TRIX.ASCIIBUF[2],2);
  WRITEITEMBLOCK(0);
END;    (* save ptrs *)

{---------------------------------------------------------------------------}

(* loads block # & byte ptr of free space always the end of the last text     *)
(* entered puts the free block number in CURRBLOCK and byte # in CURRFREEPTR. *)
PROCEDURE LOADPTRS;
VAR TRIX1 : RECORD CASE INTEGER OF
             1 : (TWOBYTES : PACKED ARRAY[0..1] OF CHAR);
             2 : (INTVALUE : INTEGER);
           END;
BEGIN
  READITEMBLOCK(0);
  MOVELEFT(TRIX.ASCIIBUF[0],TRIX1.TWOBYTES[0],2);
  CURRBLOCK := TRIX1.INTVALUE;
  MOVELEFT(TRIX.ASCIIBUF[2],TRIX1.TWOBYTES[0],2);
  CURRFREEPTR := TRIX1.INTVALUE;
END;  (* load ptrs *)
```

```
(*************************************************************************)
(*                                                                       *)
(*      Textfile : TMGR.DIR/T.GET1.TEXT           Volume : TFILES        *)
(*      Codefile : T.MGR.CODE ('Include' file)    Volume : CATDATA       *)
(*                                                                       *)
(*************************************************************************)
(* File last modified : May 20, 1983              NPRDC                  *)
(*************************************************************************)

(* list item text and/or item parameters *)
SEGMENT PROCEDURE LISTITEMS;

TYPE DINFO = RECORD
               ICODE,
               ISLOT : INTEGER;
             END;

VAR BYTECNT,
    BLKCNT,
    SAVENUM,
    STARTNUM,
    STOPNUM,
    SLOTNUM,
    SCREENBYTES,
    BLK,
    BLKPTR,
    I,
    K,
    WASTETIME,
    DATASLOT : INTEGER;

    BREAKFILE,
    TOFILE,
    DONELIST,
    CONSOLE,
    CONTINUE,
    COMPLETE,
    LISTTEXT : BOOLEAN;

    SELECT,
    OPT,
    COMMAND,
    LCOMMAND : CHAR;


    A : ARRAY[0..294] OF DINFO;

    {.................................................................}

    (* sort the directory *)
    PROCEDURE QUICKSORT;
    VAR B : INTEGER;

        {.............................................................}

        PROCEDURE SORT(L,R : INTEGER);
        VAR I,J : INTEGER;
            X,W : DINFO;
        BEGIN (* sort *)
          write('.');
          I := L;
          J := R;
          X := A[(L+R) DIV 2];
          REPEAT
            WHILE A[I].ICODE < X.ICODE DO I := I + 1;
            WHILE X.ICODE < A[J].ICODE DO J := J - 1;
            IF I <= J THEN
            BEGIN {1}
              W := A[I];
              A[I] := A[J];
              A[J] := W;
              I := I + 1;
              J := J - 1;
        END;  {1}
      UNTIL I > J;
      IF L < J THEN SORT(L,J);
      IF I < R THEN SORT(I,R);
    END;    (* sort *)

{.......................................................................}

BEGIN   (* quicksort *)
  FOR B := 0 TO 294 DO
    BEGIN {1}
      A[B].ICODE := DIRECTORY.ITEMCODE[B+6];
      A[B].ISLOT := B+6;
    END;  {1}
  SORT(0,294);
END;   (* quicksort *)

{-----------------------------------------------------------------------}

(* reads item text file & displays item text to printer or file *)
PROCEDURE LISTPRINT(BLOCKNUM, BLOCKPTR : INTEGER);
VAR X,
    Y,
    OLDY,
    CURRPTR,
    CURRBLK,
    SCREENBYTES,
    CHARCODE,
    SKIPBYTE : INTEGER;

    {...................................................................}

    (* returns next code in file *)
    FUNCTION LBUFCODE : INTEGER;
    BEGIN
      LBUFCODE := TRIX.ASCIIBUF[CURRPTR];
      CURRPTR := CURRPTR + 1;
      IF CURRPTR > 2047 THEN
      BEGIN {1}
        CURRBLK := CURRBLK + 4;
        READITEMBLOCK(CURRBLK);
        CURRPTR := 0;
      END;  {1}
    END;  (* lbufcode *)

{.......................................................................}

BEGIN  (*  listprint *)
  SCREENBYTES := (XSCREEN + 1) * (YSCREEN + 1);
  FILLCHAR(SCREEN[0],SCREENBYTES,' ');
  OLDY := TOPMAX;
  READITEMBLOCK(BLOCKNUM);
  CURRPTR := BLOCKPTR;
  CURRBLK := BLOCKNUM;
  REPEAT
    CHARCODE := LBUFCODE;
    CASE CHARCODE OF
      GOTOFLAG : BEGIN {1}
                   X := LBUFCODE;
                   Y := LBUFCODE;
                   (* ignore next byte, due to a file modification *)
                   SKIPBYTE := LBUFCODE;

                   (*
                   WHILE OLDY < Y DO
                   BEGIN {2}
                    *)
                     WRITELN(DEST);
                     (*
                     OLDY := OLDY + 1;
                   END;  {2}
                    *)
                   WRITE(DEST,' ' : X);
```

```
                END;  {1}
      PAGEFLAG : BEGIN {3}
                  WRITELN(DEST);
                  WRITELN(DEST);
                  WRITELN(DEST);
                  OLDY := TOPMAX;
                END;  {3}
      ENDITEM  : ;
    END; (* cases *)
    IF (CHARCODE >= 32) AND (CHARCODE <= 126) THEN
    BEGIN {4}
      SCREEN[X,Y] := CHR(CHARCODE);
      X := X + 1;
      WRITE(DEST,CHR(CHARCODE));
      BYTECNT := BYTECNT + 1;
    END;  {4}
  UNTIL CHARCODE = ENDITEM;
END;  (* print *)

{-----------------------------------------------------------------------}

(* lists things to the console *)
PROCEDURE LCONSOLE;
VAR FIXCHAR : CHAR;
    GRAF : BOOLEAN;
BEGIN
  IF LISTTEXT THEN
  BEGIN {1}
    GRAF := FALSE;
    IF ITEMINFO.GRAPHICS THEN
    BEGIN {2}
      FILLPORT;
      GDECODEPRINT(CURRINDEXRECNUM,A[SLOTNUM].ICODE);
      PAGE(OUTPUT);
      GRAF := TRUE;
    END   {2}
    ELSE
      DECODEPRINT(BLK,BLKPTR);
    GOTOXY(0,20);GOTOXY(20,23);
    IF SLOTNUM > MAXSAMPLES THEN
    BEGIN {3}
      IF NOT GRAF THEN
      BEGIN {4}
        IF PCRECNUM THEN GOTOXY(0,23)
        ELSE GOTOXY(0,20);
        WRITE('Item code : ',A[SLOTNUM].ICODE);
        IF PCRECNUM THEN GOTOXY(20,23)
        ELSE GOTOXY(0,21);
        WRITE('Answer :');
        IF PCRECNUM THEN
        CASE ITEMINFO.ATYPE OF
          CHARVALUE : WRITE(' ',ITEMINFO.ANSWER);
          INTVALUE  : WRITE(' ',ITEMINFO.INTANSWER);
          SEVENCHR  : BEGIN
                        FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                          WRITE(' ',ITEMINFO.CHRANSWER[I]);
                      END;
        END  (* cases *)
        ELSE
        CASE ITEMINFO.ATYPE OF
          CHARVALUE : WRITELN(' ',ITEMINFO.ANSWER);
          INTVALUE  : WRITELN(' ',ITEMINFO.INTANSWER);
          SEVENCHR  : BEGIN
                        FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                          WRITE(' ',ITEMINFO.CHRANSWER[I]);
                          WRITELN;
                      END;
        END;  (* cases *)
        IF PCRECNUM THEN GOTOXY(32,23)
        ELSE GOTOXY(0,23);
        WRITE('Press <RET> to continue, <ESC> to quit ');
        IF GETCHAR([CHR(RET),CHR(ESC)],TRUE,TRUE,TRUE) = CHR(ESC) THEN
        BEGIN {5}
```

```
              CLOSE(FILEITEMINFO,LOCK);
              EXIT(LISTITEMS);
            END;  {5}
          END   {4}
          ELSE
          BEGIN {6}
            GGOTOXY(0,20);
            GWRITESTR('Item code : ');
            GWRITEINT(A[SLOTNUM].ICODE);
            GGOTOXY(0,21);
            GWRITESTR('Answer : ');
            CASE ITEMINFO.ATYPE OF
              CHARVALUE : GWRITECHR(ITEMINFO.ANSWER);
              INTVALUE  : GWRITEINT(ITEMINFO.INTANSWER);
              SEVENCHR  : BEGIN
                            FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                            BEGIN
                              GWRITECHR(ITEMINFO.CHRANSWER[I]);
                              GWRITECHR(' ');
                            END;
                          END;
            END;  (* cases *)

            GGOTOXY(0,23);
            GWRITESTR('Press <RET> to continue, <ESC> to quit ');
            IF GETCHAR([CHR(RET),CHR(ESC)],TRUE,TRUE,TRUE) = CHR(ESC) THEN
            BEGIN {7}
              TEXTON;
              CLOSE(FILEITEMINFO,LOCK);
              EXIT(LISTITEMS);
            END;  {7}
            TEXTON;
          END;  {6}
        END   {3}
        ELSE
          GSTALL;
      END   {1}
      ELSE
      BEGIN {8}
        K := K + 1;
        WRITE(A[SLOTNUM].ICODE : 7,
                ITEMINFO.A : 18 :5,
                ITEMINFO.B : 12 :5,
                ITEMINFO.C : 11 :5);
        write('       ');
        IF ITEMINFO.GRAPHICS THEN
          WRITE('     on ')
        ELSE
          WRITE('     off');

        If ITEMINFO.DUMMY1 = 1.0 THEN
          WRITELN('      yes')
        ELSE
          WRITELN('      no');

          IF K > 18 THEN
        BEGIN {9}
          WRITELN;
          STALL;
          K := 0;
          PAGE(OUTPUT);
          WRITELN;
          WRITELN(
' Item code            A           B          C         graphics  krunched');
          WRITELN;
        END;  {9}
      END;  {8}
    END; (* lconsole *)

    {-----------------------------------------------------------------------}

    (* lists item text and data to file/printer *)
    PROCEDURE LFILE;
```

```
BEGIN
  IF LISTTEXT THEN
  BEGIN {1}
    IF SLOTNUM > MAXSAMPLES THEN
    BEGIN {2}
      WRITELN(DEST,' Item code : ',A[SLOTNUM].ICODE);
      WRITELN(DEST);
      WRITELN(DEST,' A parameter : ',ITEMINFO.A);
      WRITELN(DEST,' B parameter : ',ITEMINFO.B);
      WRITELN(DEST,' C parameter : ',ITEMINFO.C);
    END;  {2}
    LISTPRINT(BLK,BLKPTR);
    WRITELN(DEST);
    WRITELN(DEST);
    IF SLOTNUM > MAXSAMPLES THEN
    BEGIN {3}
      WRITE(DEST,' Answer(s) : ');
      CASE ITEMINFO.ATYPE OF
        CHARVALUE : WRITELN(DEST,ITEMINFO.ANSWER);
        INTVALUE  : WRITELN(DEST,ITEMINFO.INTANSWER);
        SEVENCHR  : BEGIN {4}
                      FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                        WRITE(DEST,ITEMINFO.CHRANSWER[I],' ');
                      WRITELN(DEST);
                    END;  {4}
      END;  (* cases *)
      WRITELN(DEST);
    END;  {3}
    FOR I := 1 TO 80 DO
      WRITE(DEST,'-');
    WRITELN(DEST);WRITELN(DEST);
    BYTECNT := BYTECNT + 150;
  END   {1}
  ELSE
  BEGIN {5}
    WRITE(DEST,A[SLOTNUM].ICODE : 6,
              ITEMINFO.A : 10 :4,
              ITEMINFO.B : 10 :4,
              ITEMINFO.C : 10 :4);

    IF ITEMINFO.GRAPHICS THEN
      WRITE(dest,'     on ')
    ELSE
      WRITE(dest,'     off');

    if ITEMINFO.DUMMY1 = 1.0 THEN
      WRITELN(dest,'     yes')
    ELSE
      WRITELN(dest,'     no');
  END;  {5}
END;  (* lfile *)

 {-------------------------------------------------------------------------}

PROCEDURE GETLISTINFO;
BEGIN
  PAGE(OUTPUT);
  GOTOXY(0,4);
  WRITE('Select one of the following options by entering its number.');
  GOTOXY(16,8);
  WRITE('1.  QUIT');
  GOTOXY(16,9);
  WRITE('2.  COMPLETE LISTING');
  GOTOXY(16,10);
  WRITE('3.  PARTIAL LISTING');
  GOTOXY(16,18);
  WRITE('Enter Choice # : ');
  OPT :=  GETCHAR(['1'..'3'],TRUE,TRUE,TRUE);
  CASE OPT OF
    '1' : EXIT(LISTITEMS);
    '2' : COMPLETE := TRUE;
    '3' : COMPLETE := FALSE;
  END;
```

```
      IF NOT COMPLETE THEN
      BEGIN {1}
        PAGE(OUTPUT);
        GOTOXY(20,0);
        WRITE('ENTER RANGE OF ITEMCODES TO LIST');
        WRITELN;
        WRITELN;
        WRITE('List all items between itemcode : ');
        READLN(STARTNUM);
        WRITELN;
        WRITE('And  itemcode : ');
        READLN(STOPNUM);
        IF STARTNUM > STOPNUM THEN
        BEGIN {2}
          SAVENUM := STARTNUM;
          STARTNUM := STOPNUM;
          STOPNUM := SAVENUM;
        END;  {2}
      END;  {1}

      PAGE(OUTPUT);
      GOTOXY(20,0);
      WRITE('OUTPUT SELECT MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following options by entering its number.');
      GOTOXY(16,8);
      WRITE('1.  QUIT');
      GOTOXY(16,9);
      WRITE('2.  TEXT TO CONSOLE');
      GOTOXY(16,10);
      WRITE('3.  DATA TO CONSOLE');
      GOTOXY(16,11);
      WRITE('4.  TEXT TO PRINTER');
      GOTOXY(16,12);
      WRITE('5.  DATA TO PRINTER');
      GOTOXY(16,13);
      WRITE('6.  TEXT TO FILE');
      GOTOXY(16,14);
      WRITE('7.  DATA TO FILE');
      GOTOXY(16,18);
      WRITE('Enter Choice # : ');
      SELECT :=  GETCHAR(['1'..'7'],TRUE,TRUE,TRUE);
      CONSOLE := FALSE;
      LISTTEXT := TRUE;
      IF SELECT = '1' THEN
      begin
        updatedirectory(currindexrecnum);
        EXIT(LISTITEMS);
      end;
    END;   (* get listinfo *)

{..........................................................................}

BEGIN  (* listitems *)
  GETLISTINFO;
  PAGE(OUTPUT);
  WRITE('Please wait');
  QUICKSORT;
  TOFILE := FALSE;
  CASE SELECT OF
    '1' : ;
    '2' : CONSOLE := TRUE;
    '3' : BEGIN {1}
            CONSOLE := TRUE;
            LISTTEXT := FALSE;
          END;  {1}
    '4' : REWRITE(DEST,UNITNUMPRINTER);
    '5' : BEGIN {2}
            REWRITE(DEST,UNITNUMPRINTER);
            LISTTEXT := FALSE;
          END;  {2}
    '6' : BEGIN
```

```
      GETNEWFILE;
      PAGE(OUTPUT);
      WRITE('Do you want to break the files up?  Y/N ');
      IF GETCHAR(['Y','N','y','n'],TRUE,TRUE,TRUE) IN ['Y','y'] THEN
        BREAKFILE := TRUE
      ELSE
        BREAKFILE := FALSE;
     END;
  '7' : BEGIN {3}
      LISTTEXT := FALSE;
      GETNEWFILE;
     END;  {3}
 END;

 IF SELECT IN ['6','7'] THEN
 BEGIN {4}
  TOFILE := TRUE;
  PAGE(OUTPUT);
  WRITE('Writing to file.');
 END;  {4}

 IF NOT CONSOLE THEN
 BEGIN {5}
  WRITELN(DEST);
  FOR I := 1 TO 80 DO
   WRITE(DEST,'*');
  WRITELN(DEST);
  WRITELN(DEST,DIRECTORY.TESTNAME);
  FOR I := 1 TO 80 DO
   WRITE(DEST,'*');
  WRITELN(DEST);WRITELN(DEST);
 END;  {5}

 RESET(FILEITEMINFO,DATANAME);
 K := 0;
 IF NOT LISTTEXT THEN
 BEGIN {6}
  IF CONSOLE THEN
  BEGIN {7}
   PAGE(OUTPUT);
   WRITELN(
' Item code            A           B          C          graphics  krunched');
   WRITELN;
  END   {7}
  ELSE
  BEGIN {8}
   WRITELN(DEST,
' Item code   A         B        C    graphics  krunched');
   WRITELN(DEST);
  END;  {8}
 END;  {6}

 PCRECNUM := (CURRINDEXRECNUM=4) OR (CURRINDEXRECNUM=6);
 SLOTNUM := 0;
 BYTECNT := 0;
 BLKCNT := 4;
 DONELIST := FALSE;
 REPEAT
  CONTINUE := TRUE;
  IF NOT COMPLETE THEN
   IF (A[SLOTNUM].ICODE < STARTNUM) THEN
    CONTINUE := FALSE;

  IF NOT COMPLETE THEN
   IF (A[SLOTNUM].ICODE > STOPNUM) THEN
  BEGIN {9}
   CONTINUE := FALSE;
   DONELIST := TRUE;
  END;  {9}

  IF (A[SLOTNUM].ICODE >= 0) AND (CONTINUE)  THEN
  BEGIN {10}
   IF SELECT IN ['6','7'] THEN
```

```
        WRITE('.');
      DATASLOT := HASH(A[SLOTNUM].ISLOT);
      SEEK(FILEITEMINFO,DATASLOT);
      GET(FILEITEMINFO);
      ITEMINFO := FILEITEMINFO^;
      BLK := ITEMINFO.ITEMBLOCK;
      BLKPTR := ITEMINFO.ITEMPTR;
      IF CONSOLE THEN
        LCONSOLE
      ELSE
        LFILE;

      IF BYTECNT > 512 THEN
      BEGIN
        BLKCNT := BLKCNT + 1;
        BYTECNT := BYTECNT MOD 512;
      END;

      IF (BREAKFILE) AND (BLKCNT >= 30) AND (LISTTEXT) AND (TOFILE)  THEN
      BEGIN
        CLOSE(DEST,LOCK);
        GETNEWFILE;
        BLKCNT := 4;
        BYTECNT := 0;
      END;
    END;  {10}
    SLOTNUM := SLOTNUM + 1;

  UNTIL (SLOTNUM > 294) OR DONELIST;

  IF NOT (CONSOLE) THEN
    CLOSE(DEST,LOCK)
  ELSE
    IF (NOT LISTTEXT) AND (K <> 0) THEN
    BEGIN {11}
      WRITELN;
      STALL;
    END;  {11}
  CLOSE(FILEITEMINFO,LOCK);
END;  (* listitems *)
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : TMGR.DIR/T.GET2.TEXT         Volume : TFILES               *)
(*      Codefile : T.MGR.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(* File last modified : May 25, 1983            NPROC                         *)
(******************************************************************************)


(* get test item/display text/data, allow changes *)
SEGMENT PROCEDURE FETCHITEM;
VAR DATASLOT,
    SLOT,
    BLOCK,
    BLOCKPTR,
    CODENUM : INTEGER;
    CODEOK : BOOLEAN;
    COMMAND : CHAR;
    {..........................................................................}

    (* display fetch item menu *)
    PROCEDURE ITEMMENU;
    BEGIN
      TEXT80MODE;
      NORMAL;
      GOTOXY(20,0);
      WRITE('FETCH ITEM MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following options by entering ite number.');
      GOTOXY(0,6);
      WRITE('Currently managing question : ',CODENUM);
      GOTOXY(16,10);
      WRITE('1.  QUIT');
      GOTOXY(16,11);
      WRITE('2.  DISPLAY QUESTION TEXT');
      GOTOXY(16,12);
      WRITE('3.  DISPLAY QUESTION DATA');
      GOTOXY(16,13);
      WRITE('4.  MODIFY QUESTION');
      GOTOXY(16,14);
      WRITE('5.  DELETE QUESTION');
      GOTOXY(16,15);
      WRITE('6.  SET GRAPHICS FLAG');
      GOTOXY(16,18);
      WRITE('Enter Choice # : ');
    END;  (* itemmenu *)

    {--------------------------------------------------------------------------}

    PROCEDURE SHOWINFO;
    VAR I : INTEGER;
    BEGIN
      TEXT80MODE;
      GOTOXY(22,0);
      WRITE('QUESTION DATA');
      (* display item data *)
      GOTOXY(0,3);
      WRITELN('A Parameter : ',ITEMINFO.A:5:3);
      WRITELN('B Parameter : ',ITEMINFO.B:5:3);
      WRITELN('C Parameter : ',ITEMINFO.C:5:3);
      (* show answer range and answer *)
      WRITE('Answer type : ');
      CASE ITEMINFO.ATYPE OF
        CHARVALUE : WRITELN('MULTIPLE CHOICE/SINGLE ANSWER');
        INTVALUE  : WRITELN('INTEGER VALUE ANSWER');
        SEVENCHR  : WRITELN('MULTIPLE CHOICE/MULTIPLE ANSWER [',
                    ITEMINFO.ANSWERCOUNT,' answers]');
      END;  (* cases *)
      IF ITEMINFO.ATYPE <> INTVALUE THEN
        WRITELN('Answer range : ',ITEMINFO.LOWANSWER,
                '..', ITEMINFO.HIGHANSWER);
      WRITE('Answer : ');
```

```
  CASE ITEMINFO.ATYPE OF
    CHARVALUE : WRITELN(ITEMINFO.ANSWER);
    INTVALUE  : WRITELN(ITEMINFO.INTANSWER);
    SEVENCHR  : BEGIN {1}
                  FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                    WRITE(ITEMINFO.CHRANSWER[I],' ');
                  WRITELN;
                END;  {1}
  END;  (* cases *)
  WRITE('Graphics Flag : ');
  IF NOT ITEMINFO.GRAPHICS THEN
    WRITELN('OFF')
  ELSE
    IF ITEMINFO.DUMMY1 = COMPRESSED THEN
      WRITELN('COMPRESSED ON')
    ELSE
      WRITELN('FOTOFILE ON');
  WRITELN('Block which text begins : ',ITEMINFO.ITEMBLOCK);
  WRITELN('Byte in block : ',ITEMINFO.ITEMPTR);
  WRITELN;
  WRITELN;
  STALL;
END;   (* showinfo *)

 {-----------------------------------------------------------------------}

(* displays item text and data *)
PROCEDURE SHOWITEM;
VAR I : INTEGER;
BEGIN
  IF ITEMINFO.GRAPHICS THEN
  BEGIN {1}
    FILLPORT;
    GDECODEPRINT(CURRINDEXRECNUM,DIRECTORY.ITEMCODE[SLOT]);
    PAGE(OUTPUT);
    GGOTOXY(0,20);
    GWRITESTR('Item code : ');
    GWRITEINT(DIRECTORY.ITEMCODE[SLOT]);
    GGOTOXY(0,21);
    GWRITESTR('Answer : ');
    CASE ITEMINFO.ATYPE OF
      CHARVALUE : GWRITECHR(ITEMINFO.ANSWER);
      INTVALUE  : GWRITEINT(ITEMINFO.INTANSWER);
      SEVENCHR  : BEGIN {2}
                    FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                    BEGIN {3}
                      GWRITECHR(ITEMINFO.CHRANSWER[I]);
                      GWRITECHR(' ');
                    END;  {3}
                  END;  {2}
    END;  (* cases *)

    GGOTOXY(0,23);
    GSTALL;
    TEXTON;
  END   {1}
  ELSE
  BEGIN {4}

    (* print the text *)
    DECODEPRINT(BLOCK,BLOCKPTR);

    (* show item code *)
    IF PCRECNUM THEN
    GOTOXY(0,23)
    ELSE
    GOTOXY(0,20);
    WRITE('Item code : ',DIRECTORY.ITEMCODE[SLOT]);

    IF PCRECNUM THEN
    GOTOXY(25,23)
    ELSE
    GOTOXY(0,21);
```

```
        WRITE('Answer : ');
        IF PCRECNUM THEN
        CASE ITEMINFO.ATYPE OF
          CHARVALUE : WRITE(ITEMINFO.ANSWER);
          INTVALUE  : WRITE(ITEMINFO.INTANSWER);
          SEVENCHR  : BEGIN {50}
                        FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                          WRITE(ITEMINFO.CHRANSWER[I],' ');
                      END;  {50}
        END  (* cases *)
        ELSE
        CASE ITEMINFO.ATYPE OF
          CHARVALUE : WRITELN(ITEMINFO.ANSWER);
          INTVALUE  : WRITELN(ITEMINFO.INTANSWER);
          SEVENCHR  : BEGIN {5}
                        FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                          WRITE(ITEMINFO.CHRANSWER[I],' ');
                        WRITELN;
                      END;  {5}
        END;  (* cases *)
        GOTOXY(0,23);
        IF PCRECNUM THEN
        GOTOXY(45,23);
        STALL;
      END;  {4}
    END;  (* showitem *)

    {-----------------------------------------------------------------------}

    (* delete a question from the directory *)
    PROCEDURE DELETEITEM;
    VAR I,Z  : INTEGER;
    BEGIN

      (* verify actions *)
      PAGE(OUTPUT);
      GOTOXY(16,2);
      WRITE('*********** WARNING ***********');
      WRITELN;
      WRITELN;
      WRITE('You have selected the delete question option.');
      WRITELN('  This will remove the qustion');
      WRITELN('from the subtest itempool.  Are you sure you want to delete ',
              DIRECTORY.ITEMCODE[SLOT],' ?');

      GOTOXY(16,8);
      WRITE('Press ''N'' or ''Y'' : ');
      IF GETCHAR(['Y','N','y','n'],TRUE,FALSE,TRUE) IN ['Y','y'] THEN
      BEGIN {1}
        WRITELN;
        WRITELN;

        (* display something happening *)
        WRITE('Deleting');
        FOR I := 1 TO 500 DO
          IF I MOD 50 = 0 THEN WRITE('.');

        (* mark slot as unused *)
        DIRECTORY.ITEMCODE[SLOT] := NIL;

        (* update the directory to file *)
        UPDATEDIRECTORY(CURRINDEXRECNUM);

        Z := DIRINFO[CURRINDEXRECNUM].ITEMCOUNT;
        DIRINFO[CURRINDEXRECNUM].ITEMCOUNT := Z - 1;

        PAGE(OUTPUT);
        GOTOXY(17,0);
        WRITELN('*********    IMPORTANT     *********');
        WRITELN;
        WRITELN(
'Since you have deleted a question from the subtest database, this may ');
        WRITELN(
'affect the strategy database structures.  The strategy database may contain');
        WRITELN(
'the item just deleted and if so, errors may occur during test administration');
        WRITELN(
'when the strategy selects a question not in the itempool.  Thus to make sure');
        WRITELN(
'your strategy data is still OK, you should run the ''Verify'' option for');
        WRITELN(
'each strategy data structure this subtest has.');
        WRITELN;
        WRITELN;
        STALL;
        EXIT(FETCHITEM);
      END;  {1}
    END;  (* delete *)

    {---------------------------------------------------------------------------}

    (* modify different things about a question *)
    PROCEDURE MODIFYITEM;
    VAR COMMAND : CHAR;

        {.......................................................................}

        (* display option menu *)
        PROCEDURE MODMENU;
        BEGIN
          TEXT80MODE;
          NORMAL;
          GOTOXY(20,0);
          WRITE('MODIFY QUESTION MENU');
          GOTOXY(0,4);
          WRITE('Select one of the following options by entering its number.');
          GOTOXY(0,6);
          WRITE('Currently modifying question ',CODENUM);
          GOTOXY(16,10);
          WRITE('1.  QUIT');
          GOTOXY(16,11);
          WRITE('2.  CHANGE QUESTION CODE');
          GOTOXY(16,12);
          WRITE('3.  CHANGE QUESTION TEXT');
          GOTOXY(16,13);
          WRITE('4.  CHANGE ANSWER RANGE');
          GOTOXY(16,14);
          WRITE('5.  CHANGE ANSWER');
          GOTOXY(16,15);
          WRITE('6.  CHANGE QUESTION DATA');
          GOTOXY(16,19);
          WRITE('Enter Choice # : ');
        END;   (* modmenu *)

        {-----------------------------------------------------------------------}

        (* allows change of id codes *)
        PROCEDURE GETNEWID;
        VAR NEWID,
            CHECKSLOT : INTEGER;
            CODEOK : BOOLEAN;
        BEGIN
          PAGE(OUTPUT);
          WRITELN;
          WRITELN('The current question id code is : ',
                   DIRECTORY.ITEMCODE[SLOT]);
          CODEOK := FALSE;
          REPEAT  (* get code not previously used *)
            WRITELN;
            WRITE('Enter the new identification code and then press <RET> : ');
              READLN(NEWID);
              IF NEWID >= 0 THEN
              BEGIN {1}
                DIRECTORY.ITEMCODE[SLOT] := NIL;
                CHECKSLOT := SLOTSEARCH(NEWID);
                IF CHECKSLOT <> NIL THEN
```

181

```
                BEGIN (2)
                  WRITELN;
                  WRITELN('Code # ',NEWID,' previously used !');
                  WRITELN;
                  SQUAWK;
                END   (2)
                ELSE
                  CODEOK := TRUE;
              END   (1)
              ELSE
              BEGIN (3)
                SQUAWK;
                WRITELN;
                WRITELN('Id # must be a positive number.');
                WRITELN;
              END;  (3)
          UNTIL CODEOK;
          DIRECTORY.ITEMCODE[SLOT] := NEWID;
          WRITELN;
          WRITELN('The new id # is : ',NEWID);
          CODENUM := NEWID;
          WRITELN;
          WRITELN;
          WRITELN;
          WRITELN('                    *********    IMPORTANT     *********');
          WRITELN;
          WRITELN(
'Since you have changed the question id code, this may affect the strategy');
          WRITELN(
'database structures.  The strategy database may contain the item just');
          WRITELN(
'changed and if so, errors may occur during test administration when the');
          WRITELN(
'strategy selects a question no longer identified by its old id #.  Thus to');
          WRITELN(
'make sure your strategy data is still OK, you should run the ''Verify'' ');
          WRITELN(
'option for each strategy data structure this subtest has.');
          WRITELN;
          WRITELN;
          STALL;
          UPDATEDIRECTORY(CURRINDEXRECNUM);
        END;  (* getnewid *)

        {--------------------------------------------------------------------}

        (* change the question text *)
        PROCEDURE CHANGETEXT;
        VAR MODPTR,
            MODBLOCK : INTEGER;
            ENDTEXT  : BOOLEAN;

            {..............................................................}

            (* reads item text file & displays page *)
            PROCEDURE PAGEPRINT(VAR LASTPAGE : BOOLEAN);
            VAR X,
                Y,
                SCREENBYTES,
                CHARCODE,
                  SKIPBYTE : INTEGER;

                {..........................................................}

                (* returns next  code in item  file *)
                FUNCTION BUFFERCODE : INTEGER;
                BEGIN
                  BUFFERCODE := TRIX.ASCIIBUF[MODPTR];
                  MODPTR := MODPTR + 1;
                  IF MODPTR > 2047 THEN
                  BEGIN (1)
                    MODBLOCK := MODBLOCK + 4;
                    READITEMBLOCK(MODBLOCK);
            MODPTR := 3;
          END;  (1)
        END;  (* buffercode *)

    {............................................................}

    BEGIN  (* page print *)
      SCREENBYTES := (XSCREEN + 1) * (YSCREEN + 1);
      FILLCHAR(SCREEN[0],SCREENBYTES,' ');
      SETSCREEN;
      READITEMBLOCK(MODBLOCK);
      REPEAT
        CHARCODE := BUFFERCODE;
        CASE CHARCODE OF
          GOTOFLAG : BEGIN (1)
                       X := BUFFERCODE;
                       Y := BUFFERCODE;
                       GOTOXY(X,Y);

                       (* skip next byte, this was an easy way *)
                       (* to get around the modification of the*)
                       (* ascii file, where an extra byte was  *)
                       (* used to store the # of characters per*)
                       (* line.                                *)
                       SKIPBYTE := BUFFERCODE;
                     END;  (1)
          PAGEFLAG : ;
          ENDITEM  : ;
        END;  (* cases *)
        IF (CHARCODE >= 32) AND (CHARCODE <= 126) THEN
        BEGIN (2)
          SCREEN[X,Y] := CHR(CHARCODE);
          X := X + 1;
          WRITE(CHR(CHARCODE));
        END;  (2)
      UNTIL (CHARCODE = ENDITEM) OR (CHARCODE = PAGEFLAG);
      IF CHARCODE = ENDITEM THEN
        LASTPAGE := TRUE
      ELSE
        LASTPAGE := FALSE;
    END;  (* page print *)

{............................................................}


BEGIN (* change text *)

  (* get pointers to free space in file *)
  LOADPTRS;

  (* get pointers to text being modified *)
  MODBLOCK := ITEMINFO.ITEMBLOCK;
  MODPTR := ITEMINFO.ITEMPTR;

  (* save ptrs to beginning of new modified text *)
  ITEMINFO.ITEMBLOCK := CURRBLOCK;
  ITEMINFO.ITEMPTR := CURRFREEPTR;

  BLOCK := CURRBLOCK;
  BLOCKPTR := CURRFREEPTR;

  (* change the text page by page, if page is not modified, then *)
  (* write it out as is.                                         *)
  REPEAT


    (* display the text being modified *)
    PAGEPRINT(ENDTEXT);

    (* edit the screen buffer *)
    FILLSCREENBUFFER(TOPMAX,RIGHTMAX,
                     BOTTOMMAX,LEFTMAX,FALSE);
```

184

```
      (* if it is not the end of the text then write current page *)
      (* to block buffer to file.                                 *)
      IF NOT (ENDTEXT) THEN
        CODESCREEN(FALSE);
  UNTIL ENDTEXT;

  (* write last page to buffer then to file *)
  CODESCREEN(TRUE);

  (* save location of next place to put text *)
  SAVEPTRS;
END;  (* changetext *)

{------------------------------------------------------------------}

(* change the answer selection range data *)
PROCEDURE CHANGERANGE;
VAR LBOUND,
    HBOUND : CHAR;
    ERR : BOOLEAN;
BEGIN

  (* display the question *)
  DECODEPRINT(BLOCK,BLOCKPTR);

  (* get new answer range *)
  GOTOXY(0,20);
  WRITELN('Current range : ',ITEMINFO.LOWANSWER,'..',
           ITEMINFO.HIGHANSWER);
  WRITE('New low bound?  (letter/digit) : ');
  LBOUND := GETCHAR(['0'..'9','A'..'Z'],TRUE,TRUE,TRUE);
  WRITELN;
  WRITE('New high bound?  (letter/digit) : ');
  HBOUND := GETCHAR(['0'..'9','A'..'Z'],TRUE,TRUE,TRUE);
  ERR := FALSE;
  IF (HBOUND IN ['0'..'9']) THEN
    IF NOT (LBOUND IN ['0'..'9']) THEN
      ERR := TRUE;
  IF (HBOUND IN ['A'..'Z']) THEN
    IF NOT (LBOUND IN ['A'..'Z']) THEN
      ERR := TRUE;
  IF ERR THEN
  BEGIN {1}
    BLANKLINES(20,4,20);
    WRITELN('Bounds range type mismatch error!');
    SQUAWK;
    WRITELN;
    STALL;
    EXIT(CHANGERANGE);
  END;  {1}

  IF ORD(LBOUND) > ORD(HBOUND) THEN
  BEGIN {2}
    BLANKLINES(20,4,20);
    WRITELN('Low bound exceeds high bound error!');
    SQUAWK;
    WRITELN;
    STALL;
    EXIT(CHANGERANGE);
  END;  {2}

  ITEMINFO.HIGHANSWER := HBOUND;
  ITEMINFO.LOWANSWER := LBOUND;
  BLANKLINES(20,4,20);
  WRITELN('The new range is : ',ITEMINFO.LOWANSWER,'..',
           ITEMINFO.HIGHANSWER);
  WRITELN;
  STALL;
END;  (* changerange *)

{------------------------------------------------------------------}

(* change the answer  *)
```

```
PROCEDURE CHANGEANSWER;
VAR BOUND,
    SELECT : CHAR;
    I : INTEGER;

    {...........................................................}

    (* get the new answer *)
    PROCEDURE GETANSWER;
    BEGIN
      SELECT := GETCHAR(['1'..'3'],TRUE,TRUE,TRUE);
      PAGE(OUTPUT);
      CASE SELECT OF
       '1' : BEGIN {1}
               GOTOXY(22,0);
               WRITE('MULTIPLE CHOICE/SINGLE ANSWER');
               ITEMINFO.ATYPE := CHARVALUE;
               GOTOXY(0,6);
               WRITE('Enter new answer and then press <RET> : ');
               FILLBUF(1,[ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER],
                       TRUE);
               ITEMINFO.ANSWER := LINEBUF[0];
               LINEBUF[0] := ' ';
             END;  {1}
       '2' : BEGIN {2}
               GOTOXY(22,0);
               WRITE('INTEGER VALUE ANSWER');
               ITEMINFO.ATYPE := INTVALUE;
               GOTOXY(0,6);
               WRITE('Enter new answer (integer ',
                     'value) and then press <RET> : ');
               READLN(I);
               ITEMINFO.INTANSWER := I;
             END;  {2}
       '3' : BEGIN {3}
               GOTOXY(22,0);
               WRITE('MULTIPLE CHOICE/MULTIPLE ANSWERS');
               ITEMINFO.ATYPE := SEVENCHR;
               GOTOXY(0,6);
               WRITE('Enter the number of answers',
                     ' this question has (1..7) : ');
               SELECT := GETCHAR(['1'..'7'],TRUE,TRUE,TRUE);
               ITEMINFO.ANSWERCOUNT := ORD(SELECT) - 48;
               WRITELN;
               WRITELN;
               FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
               BEGIN {4}
                 WRITE('Enter answer ',I,', then press <RET> : ');
                 FILLBUF(1,
                         [ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER],
                         TRUE);
                 ITEMINFO.CHRANSWER[I] := LINEBUF[0];
                 LINEBUF[0] := ' ';
                 WRITELN;
               END;  {4}
             END;  {3}
      END;  (* cases *)
    END;  (* get answer *)

{.......................................................................}

BEGIN  (* change answer *)
  DECODEPRINT(BLOCK,BLOCKPTR);
  GOTOXY(0,20);
  WRITE('Current answer(s) : ');
  CASE ITEMINFO.ATYPE OF
    CHARVALUE : WRITELN(ITEMINFO.ANSWER);
    INTVALUE  : WRITELN(ITEMINFO.INTANSWER);
    SEVENCHR  : BEGIN {1}
                  FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                    WRITE(ITEMINFO.CHRANSWER[I],' ');
                  WRITELN;
                END;  {1}
    END;  (* cases *)
    WRITELN;
    STALL;
    TEXT80MODE;
    GOTOXY(20,0);
    WRITE('SELECT ANSWER TYPE');
    GOTOXY(0,4);
    WRITE('Select one of the following options by entering its number.');
    GOTOXY(16,8);
    WRITE('1.  MULTIPLE CHOICE/SINGLE ANSWER');
    GOTOXY(16,9);
    WRITE('2.  INTEGER VALUE ANSWER');
    GOTOXY(16,10);
    WRITE('3.  MULTIPLE CHOICE/MULTIPLE ANSWERS');
    GOTOXY(16,14);
    WRITE('Enter Choice # : ');
    GETANSWER;
    WRITELN;
    WRITELN;
    WRITE('The new answer(s) is : ');
    CASE ITEMINFO.ATYPE OF
      CHARVALUE : WRITELN(ITEMINFO.ANSWER);
      INTVALUE  : WRITELN(ITEMINFO.INTANSWER);
      SEVENCHR  : BEGIN {2}
                    FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                      WRITE(ITEMINFO.CHRANSWER[I],' ');
                    WRITELN;
                  END;  {2}
    END;  (* cases *)
    WRITELN;
    STALL;
  END;   (* changeanswer *)

  {-------------------------------------------------------------------}

  (* change question data *)
  PROCEDURE CHANGEDATA;
  VAR SELECT : CHAR;
  BEGIN
    REPEAT
      PAGE(OUTPUT);
      GOTOXY(20,0);
      WRITE('CHANGE QUESTION DATA MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following options',
            ' by entering its number.');
      GOTOXY(0,6);
      WRITE('Current data : ');
      GOTOXY(15,6);
      WRITE('A parameter = ',ITEMINFO.A:5:3);
      GOTOXY(15,7);
      WRITE('B parameter = ',ITEMINFO.B:5:3);
      GOTOXY(15,8);
      WRITE('C parameter = ',ITEMINFO.C:5:3);
      GOTOXY(16,12);
      WRITE('1.  QUIT');
      GOTOXY(16,13);
      WRITE('2.  CHANGE PARAMETER A');
      GOTOXY(16,14);
      WRITE('3.  CHANGE PARAMETER B');
      GOTOXY(16,15);
      WRITE('4.  CHANGE PARAMETER C');
      GOTOXY(16,18);
      WRITE('Enter Choice # : ');
      SELECT :=  GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
      CASE SELECT OF
        '1' : ;
        '2' : BEGIN {1}
                GETPARAM('A');
                ITEMINFO.A := AVALUE;
              END;  {1}
        '3' : BEGIN {2}
                GETPARAM('B');
```

```
                      ITEMINFO.B := BVALUE;
                    END;  {2}
              '4' : BEGIN {3}
                      GETPARAM('C');
                      ITEMINFO.C := CVALUE;
                    END;  {3}
            END; (* cases *)
          UNTIL SELECT = '1';
        END;  (* changedata *)

    {.....................................................................}

    BEGIN  (* modify *)
      REPEAT
        MODMENU;
        COMMAND := GETCHAR(['1'..'6'],TRUE,TRUE,TRUE);
        CASE COMMAND OF
          '1' : ;
          '2' : GETNEWID;
          '3' : CHANGETEXT;
          '4' : CHANGERANGE;
          '5' : CHANGEANSWER;
          '6' : CHANGEDATA;
        END; (* cases *)
      UNTIL COMMAND = '1';

      (* update data record to file *)
      UPDATEITEMFILE(DATASLOT);
    END;  (* modify *)

{.........................................................................}

BEGIN  (* fetch item *)
  PAGE(OUTPUT);

 PCRECNUM := (CURRINDEXRECNUM = 4) OR (CURRINDEXRECNUM = 6);
   (* is current test PC or PC-2 *)

  (* get item code to fetch *)
  CODEOK := FALSE;
  REPEAT
    WRITE('Enter the question code, then press <RET> : ');
  (*$I-*)
    READLN(CODENUM);
  (*$I+*)

    (* if valid code # >= 0 *)
    IF CODENUM >= 0 THEN
    BEGIN {1}
      SLOT := SLOTSEARCH(CODENUM);

      (* if code # does not exist in subtest directory *)
      IF SLOT = NIL THEN
      BEGIN {2}
        WRITELN;
        WRITELN('No item with code # ',CODENUM);
        SQUAWK;
        WRITELN;
        STALL;
        EXIT(FETCHITEM);
      END   {2}
      ELSE
        CODEOK := TRUE;
    END   {1}
    ELSE
    BEGIN {3}
      SQUAWK;
      WRITELN;
      WRITELN('Id # must be a positive number.');
      WRITELN;
      STALL;
      EXIT(FETCHITEM);
    END;  {3}
  UNTIL CODEOK;

  (* get location of data *)
  DATASLOT := HASH(SLOT);
  PAGE(OUTPUT);

  (* get the data *)
  RESET(FILEITEMINFO,DATANAME);
  SEEK(FILEITEMINFO,DATASLOT);
  GET(FILEITEMINFO);
  ITEMINFO := FILEITEMINFO^;
  CLOSE(FILEITEMINFO,LOCK);
  BLOCK := ITEMINFO.ITEMBLOCK;
  BLOCKPTR := ITEMINFO.ITEMPTR;

  (* allow various options *)
  REPEAT
    ITEMMENU;
    COMMAND := GETCHAR(['1'..'6'],TRUE,TRUE,TRUE);
    CASE COMMAND OF
      '1' : ;
      '2' : SHOWITEM;
      '3' : SHOWINFO;
      '4' : MODIFYITEM;
      '5' : DELETEITEM;
      '6' : BEGIN {4}
              SETGFLAG;
              (* update data record to file *)
              UPDATEITEMFILE(DATASLOT);
            END;  {4}
    END;  (* cases *)
  UNTIL COMMAND ='1';
END;   (* fetch item *)

{--------------------------------------------------------------------------}

(* loads a test , allows various options *)
SEGMENT PROCEDURE FETCHTEST;
VAR COMMAND : CHAR;

    {..................................................................}

    (* displays command selection for fetch test *)
    PROCEDURE FETCHMENU;
    BEGIN
      TEXT80MODE;
      NORMAL;
      GOTOXY(20,0);
      WRITE('FETCH SUBTEST MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following procedures by entering its number.');
      GOTOXY(0,6);
      WRITE('Currently managing subtest : ',DIRECTORY.TESTNAME);
      GOTOXY(16,9);
      WRITE('1.  QUIT');
      GOTOXY(16,10);
      WRITE('2.  ADD NEW QUESTION');
      GOTOXY(16,11);
      WRITE('3.  FETCH A QUESTION');
      GOTOXY(16,12);
      WRITE('4.  LIST SUBTEST QUESTIONS');
      GOTOXY(16,13);
      WRITE('5.  SAMPLE QUESTION MANAGEMENT');
      GOTOXY(16,14);
      WRITE('6.  SUBTEST INSTRUCTIONS MANAGEMENT');
      GOTOXY(16,18);
      WRITE('Enter Choice # : ');
    END;

{..........................................................................}

BEGIN  (* fetchtest *)
  REPEAT
```

```
    FETCHMENU;
    COMMAND := GETCHAR(['1'..'6'],TRUE,TRUE,TRUE);
    CASE COMMAND OF
      '1' : ;
      '2' : INSERTITEM;
      '3' : FETCHITEM;
      '4' : LISTITEMS;
      '5' : SAMPLEQUESTIONS;
      '6' : INSTRUCTIONS;
    END;  (* cases *)
  UNTIL COMMAND ='1';
END;   (* fetch test *)
```

```
(**************************************************************************)
(*                                                                        *)
(*      Textfile : TMGR.DIR/T.INSTR.TEXT        Volume : TFILES           *)
(*      Codefile : T.MGR.CODE ('Include' file)  Volume : CATDATA          *)
(*                                                                        *)
(**************************************************************************)
(* File last modified : Feb 18, 1983            NPRDC                     *)
(**************************************************************************)

SEGMENT PROCEDURE INSTRUCTIONS;

VAR
  HASH_LOC   : INTEGER;
  COMMAND : CHAR;

  {.......................................................................}

  PROCEDURE MENU;
  BEGIN
    TEXT80MODE;
    NORMAL;
    GOTOXY(20,0);
    WRITE('SUBTEST INSTRUCTIONS MENU');
    GOTOXY(0,4);
    WRITE('Select one of the following procedures by entering its number.');
    GOTOXY(16,8);
    WRITE('1.  QUIT');
    GOTOXY(16,9);
    WRITE('2.  ENTER INSTRUCTIONS');
    GOTOXY(16,10);
    WRITE('3.  VIEW INSTRUCTIONS');
    GOTOXY(16,11);
    WRITE('4.  MODIFY INSTRUCTIONS');
    GOTOXY(16,12);
    WRITE('5.  DELETE INSTRUCTIONS');
    GOTOXY(16,16);
    WRITE('Enter Choice # : ');
  END;  (* menu *)

  {-----------------------------------------------------------------------}

  PROCEDURE ENTER_INSTRUCTIONS;
  BEGIN
    IF DIRECTORY.ITEMCODE[0] > 0
      THEN
          BEGIN {1}
            PAGE(OUTPUT);
            WRITELN;
            WRITELN('Subtest instructions already exist.');
            WRITELN;
            SQUAWK;
            STALL;
            EXIT(ENTER_INSTRUCTIONS);
          END;  {1}

    DIRECTORY.ITEM_CODE[0] := 1;
    LOADPTRS;
    ITEM_INFO.ITEM_PTR := CURR_FREE_PTR;
    ITEM_INFO.ITEM_BLOCK := CURR_BLOCK;
    FILL_SCREEN_BUFFER(TOP_MAX, RIGHT_MAX, BOTTOM_MAX, LEFT_MAX, TRUE);
    CODE_SCREEN(TRUE);
    SAVE_PTRS;
    UPDATE_DIRECTORY(CURR_INDEX_REC_NUM);
    HASH_LOC := HASH(0);
    UPDATE_ITEM_FILE(HASH_LOC);
  END;  (* enter instructions *)

  {-----------------------------------------------------------------------}

  PROCEDURE MODIFYINSTRUCTIONS;
  VAR MODPTR,
      MODBLOCK,
      OLDPTR,
   OLDBLOCK,
   BLOCK,
   BLOCKPTR : INTEGER;
   ENDTEXT,
   RECODE,
   DONE : BOOLEAN;

   {.....................................................................}


   (* reads item text file & displays page *)
   PROCEDURE PAGEPRINT(VAR LASTPAGE : BOOLEAN);
   VAR X,
       Y,
       SCREENBYTES,
       CHARCODE,
       SKIPBYTE  : INTEGER;

       {.............................................................}

       (* returns next  code in item  file *)
       FUNCTION BUFFERCODE : INTEGER;
       BEGIN
         BUFFERCODE := TRIX.ASCIIBUF[MODPTR];
         MODPTR := MODPTR + 1;
         IF MODPTR > 2047
           THEN
               BEGIN {1}
                 MODBLOCK := MODBLOCK + 4;
                 READITEMBLOCK(MODBLOCK);
                 MODPTR := 0;
               END;  {1}
       END;  (* buffercode *)

   {.....................................................................}

   BEGIN  (* page print *)
    SCREENBYTES :=
      (XSCREEN + 1) * (YSCREEN + 1);
    FILLCHAR(SCREEN[0],SCREENBYTES,' ');
    SETSCREEN;
    READITEMBLOCK(MODBLOCK);

    REPEAT
      CHARCODE := BUFFERCODE;
      CASE CHARCODE OF
        GOTOFLAG : BEGIN {1}
                     X := BUFFERCODE;
                     Y := BUFFERCODE;
                     SKIPBYTE := BUFFERCODE;
                     GOTOXY(X,Y);
                   END;  {1}
        PAGEFLAG : ;
        ENDITEM  : ;
      END;  (* cases *)

      IF (CHARCODE >= 32) AND (CHARCODE <= 126)
        THEN
            BEGIN {2}
              SCREEN[X,Y] := CHR(CHARCODE);
              X := X + 1;
              WRITE(CHR(CHARCODE));
            END;  {2}
    UNTIL (CHARCODE = ENDITEM) OR (CHARCODE = PAGEFLAG);

    IF CHARCODE = ENDITEM
      THEN
          LASTPAGE := TRUE
      ELSE
          LASTPAGE := FALSE;
  END;  (* page print *)

{.........................................................................}
```

```
BEGIN  (* modify instructions *)
  IF DIRECTORY.ITEMCODE[0] < 0
    THEN
      BEGIN {1}
        PAGE(OUTPUT);
        WRITELN;
        WRITELN('No instructions to modify......');
        WRITELN;
        SQUAWK;
        STALL;
        EXIT(MODIFYINSTRUCTIONS);
      END;  {1}

  HASH_LOC := HASH(0);
  RESET(FILEITEMINFO,DATANAME);
  SEEK(FILEITEMINFO,HASH_LOC);
  GET(FILEITEMINFO);
  ITEMINFO := FILEITEMINFO^;
  CLOSE(FILEITEMINFO,LOCK);
  PAGE(OUTPUT);
  WRITELN;
  WRITE('Change instruction text?  Press ''N'' or ''Y'' : ');

  IF GETCHAR(['y','n','Y','N'],TRUE,TRUE,TRUE) IN ['Y','y']
    THEN
      BEGIN {2}
        LOADPTRS;
        MODBLOCK := ITEMINFO.ITEMBLOCK;
        MODPTR := ITEMINFO.ITEMPTR;
        (* save ptrs to beginning of text *)
        ITEMINFO.ITEMBLOCK := CURRBLOCK;
        ITEMINFO.ITEMPTR := CURRFREEPTR;
        (* save old ptrs to delete old text *)
        OLDBLOCK := MODBLOCK;
        OLDPTR := MODPTR;
        BLOCK := CURRBLOCK;
        BLOCKPTR := CURRFREEPTR;
        RECODE := TRUE;

        REPEAT
          PAGEPRINT(ENDTEXT);
          FILLSCREENBUFFER(TOPMAX, RIGHTMAX, BOTTOMMAX, LEFTMAX, FALSE);
          IF NOT (ENDTEXT) THEN CODESCREEN(FALSE);
        UNTIL ENDTEXT;
      END  {2}
    ELSE
      RECODE := FALSE;

  IF RECODE
    THEN
      BEGIN {3}
        CODESCREEN(TRUE);
        SAVEPTRS;
      END;  {3}
  UPDATEITEMFILE(HASH_LOC);
END;  (* modify instructions *)

{------------------------------------------------------------------------}

PROCEDURE VIEWINSTRUCTIONS;
VAR BLOCK,
    BLOCKPTR : INTEGER;

BEGIN  (* view instructions *)
  IF DIRECTORY.ITEMCODE[0] < 0
    THEN
      BEGIN {1}
        PAGE(OUTPUT);
        WRITELN;
        WRITELN('No instructions to view......');
        WRITELN;
        SQUAWK;
```

```
            STALL;
            EXIT(VIEWINSTRUCTIONS);
          END;  (1)

    HASH_LOC := HASH(0);
    RESET(FILEITEMINFO,DATANAME);
    SEEK(FILEITEMINFO,HASH_LOC);
    GET(FILEITEMINFO);
    ITEMINFO := FILEITEMINFO^;
    CLOSE(FILEITEMINFO,LOCK);
    BLOCK := ITEMINFO.ITEMBLOCK;
    BLOCKPTR := ITEMINFO.ITEMPTR;
    DECODEPRINT(BLOCK,BLOCKPTR);
    GOTOXY(0,21);
    STALL;
  END;  (* view  instructions *)

  {----------------------------------------------------------------------------}

  PROCEDURE DELETEINSTRUCTIONS;
  VAR I,HASHLOC : INTEGER;
  BEGIN
    IF DIRECTORY.ITEMCODE[0] < 0
      THEN
          BEGIN (1)
            PAGE(OUTPUT);
            WRITELN;
            WRITELN('No instructions to delete......');
            WRITELN;
            SQUAWK;
            STALL;
            EXIT(DELETEINSTRUCTIONS);
          END;  (1)

    PAGE(OUTPUT);
    GOTOXY(15,0);
    WRITE('********** WARNING **********');
    GOTOXY(0,3);
    WRITELN('You have selected the ''delete instructions'' option.',
            ' This will');
    WRITELN('purge the existing instructions from the files.');
    WRITELN;
    WRITELN;
    WRITE('Do you wish to delete the instructions?  Press ''N'' or ''Y'' : ');

    IF GETCHAR(['Y','y','N','n'],TRUE,TRUE,TRUE) IN ['Y','y']
      THEN
          BEGIN (2)
            PAGE(OUTPUT);
            WRITELN;
            WRITE('Deleting instructions ');
            FOR I := 1 TO 500 DO
              IF I MOD 50 = 0 THEN WRITE('.');
            DIRECTORY.ITEMCODE[0] := NIL;
            UPDATEDIRECTORY(CURRINDEXRECNUM);
          END;  (2)
  END;  (* delete instructions *)

{..............................................................................}

BEGIN  (* instructions *)
  REPEAT
    MENU;
    COMMAND := GETCHAR(['1'..'5'],TRUE,FALSE,TRUE);
    CASE COMMAND OF
      '1' : ;
      '2' : ENTER_INSTRUCTIONS;
      '3' : VIEWINSTRUCTIONS;
      '4' : MODIFY_INSTRUCTIONS;
      '5' : DELETEINSTRUCTIONS;
    END;  (* cases *)
  UNTIL COMMAND = '1';
END;  (* instructions *)
```

```
(************************************************************************)
(*                                                                      *)
(*      Textfile : TMGR.DIR/T.SAMPLES.TEXT      Volume : TFILES         *)
(*      Codefile : T.MGR.CODE ('Include' file)  Volume : CATDATA        *)
(*                                                                      *)
(************************************************************************)
(*  File last modified : Feb 18, 1983           NPRDC                   *)
(************************************************************************)

SEGMENT PROCEDURE SAMPLEQUESTIONS;

VAR I,
    SCOUNT,
    HASH_LOC    : INTEGER;
    COMMAND : CHAR;

    {....................................................................}

    (* sample menu *)
    PROCEDURE MENU;
    BEGIN
      TEXT80MODE;
      NORMAL;
      GOTOXY(20,0);
      WRITE('SUBTEST SAMPLES MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following procedures by entering its number.');
      GOTOXY(16,8);
      WRITE('1.  QUIT');
      GOTOXY(16,9);
      WRITE('2.  ENTER SAMPLE QUESTION');
      GOTOXY(16,10);
      WRITE('3.  VIEW SAMPLE QUESTION');
      GOTOXY(16,11);
      WRITE('4.  MODIFY SAMPLE QUESTION');
      GOTOXY(16,12);
      WRITE('5.  DELETE SAMPLE QUESTION');
      GOTOXY(16,16);
      WRITE('Enter Choice # : ');
    END;  (* menu *)

    {--------------------------------------------------------------------}

    (* remove a sample question *)
    PROCEDURE DELETESAMPLE;
    VAR HASHLOC,
        Z       : INTEGER;
        ERROR   : BOOLEAN;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(15,0);
      WRITE('********** WARNING **********');
      GOTOXY(0,3);
      WRITELN('You have selected the ''delete sample question '' option.',
             ' This will');
      WRITELN('purge an existing sample question from the files.');
      WRITELN;
      WRITELN;
      WRITE('Do you wish to continue?  Press ''N'' or ''Y'' : ');
      IF GETCHAR(['Y','y','N','n'],TRUE,TRUE,TRUE) IN ['Y','y']
        THEN
           BEGIN {1}
             PAGE(OUTPUT);
             WRITELN;
             WRITELN('Delete which sample question?     (1..5)');
             WRITE('Enter the number then press <RET> : ');
             READLN(Z);
             ERROR := FALSE;
             IF (Z > MAXSAMPLES) OR (Z < 1)
               THEN
                   ERROR := TRUE
               ELSE
                   IF DIRECTORY.ITEMCODE[Z] < 0 THEN ERROR := TRUE;
```

```
          IF ERROR
            THEN
                BEGIN {2}
                  WRITELN;
                  WRITELN;
                  WRITELN('No such sample question!');
                  SQUAWK;
                  WRITELN;
                  STALL;
                  EXIT(DELETESAMPLE);
                END;  {2}
          DIRECTORY.ITEMCODE[Z] := NIL;
          SCOUNT := SCOUNT - 1;
          UPDATEDIRECTORY(CURRINDEXRECNUM);
          PAGE(OUTPUT);
          WRITELN;
          WRITE('Deleting sample question.');
          FOR Z := 1 TO 500 DO
            IF (Z MOD 50) = 0 THEN WRITE('.');
        END;  {1}
END;  (* delete samples *)

{-----------------------------------------------------------------------------}

(* add a sample question *)
PROCEDURE ENTER_SAMPLES;
VAR I,
    X,Z : INTEGER;
    SELECT : CHAR;

BEGIN  (* enter samples *)
  PAGE(OUTPUT);
  X := 0;
  Z := 0;
  REPEAT
    X := X + 1;
    IF DIRECTORY.ITEMCODE[X] > 0
      THEN
          BEGIN {1}
            Z := Z + 1;
            WRITELN('Sample question ',X,' exists.');
          END;  {1}
  UNTIL (X >= MAXSAMPLES);
  IF Z >= MAXSAMPLES
    THEN
        BEGIN {2}
          WRITELN;
          WRITELN('No room for more sample questions.');
          WRITELN;
          SQUAWK;
          STALL;
          EXIT(ENTERSAMPLES);
        END;  {2}
  WRITELN;
  WRITELN;
  WRITELN('Make which sample question?    (1..5)');
  WRITE('Enter the number then press <RET> : ');
  READLN(Z);
  IF (Z > MAXSAMPLES) OR (Z < 1)
    THEN
        BEGIN {3}
          WRITELN;
          WRITELN;
          WRITELN('Bad input!  Should be a number from 1 to 5.');
          SQUAWK;
          WRITELN;
          STALL;
          EXIT(ENTERSAMPLES);
        END   {3}
    ELSE
        IF DIRECTORY.ITEMCODE[Z] > 0
          THEN
              BEGIN {4}
```

```
       WRITELN;
       WRITELN;
       WRITELN('Sample question ',Z,' exists already!');
       SQUAWK;
       WRITELN;
       STALL;
       EXIT(ENTERSAMPLES);
      END;  (4)
 DIRECTORY.ITEM_CODE[Z] := 1;
 LOADPTRS;
 ITEM_INFO.ITEM_PTR := CURR_FREE_PTR;
 ITEM_INFO.ITEM_BLOCK := CURR_BLOCK;
 FILL_SCREEN_BUFFER(TOP_MAX, RIGHT_MAX, BOTTOM_MAX, LEFT_MAX, TRUE);
 TEXT80MODE;
 SETGFLAG;
 SELECTATYPE;
 CODE_SCREEN(TRUE);
 SAVE_PTRS;
 UPDATE_DIRECTORY(CURR_INDEX_REC_NUM);
 HASH_LOC := HASH(Z);
 UPDATE_ITEM_FILE(HASH_LOC);
 SCOUNT := SCOUNT + 1;
END;  (* enter samples *)

{--------------------------------------------------------------------}

(* modify an existing sample *)
PROCEDURE MODIFYSAMPLES;
VAR Z,
  I,
  MODPTR,
  MODBLOCK,
  BLOCK,
  BLOCKPTR : INTEGER;
  ENDTEXT,
  ERROR,
  DONE : BOOLEAN;
  SELECT : CHAR;

  {..............................................................}

  (* change the answer selection range data *)
  PROCEDURE CHANGERANGE;
  VAR LBOUND,
    HBOUND : CHAR;
    ERR : BOOLEAN;

  BEGIN

   (* display the question *)
   DECODEPRINT(MODBLOCK,MODPTR);

   (* get new answer range *)
   GOTOXY(0,20);
   WRITELN('Current range : ',ITEMINFO.LOWANSWER,'..',
        ITEMINFO.HIGHANSWER);
   WRITE('New low bound?  (letter/digit) : ');
   LBOUND := GETCHAR(['0'..'9','A'..'Z'],TRUE,TRUE,TRUE);
   WRITELN;
   WRITE('New high bound?  (letter/digit) : ');
   HBOUND := GETCHAR(['0'..'9','A'..'Z'],TRUE,TRUE,TRUE);
   ERR := FALSE;

   IF (HBOUND IN ['0'..'9'])
    THEN
      IF NOT (LBOUND IN ['0'..'9']) THEN ERR := TRUE;

   IF (HBOUND IN ['A'..'Z'])
    THEN
      IF NOT (LBOUND IN ['A'..'Z']) THEN ERR := TRUE;

   IF ERR
    THEN
```

```
        BEGIN {1}
          BLANKLINES(20,4,20);
          WRITELN('Bounds range type mismatch error!');
          SQUAWK;
          WRITELN;
          STALL;
          EXIT(CHANGERANGE);
        END;  {1}

  IF ORD(LBOUND) > ORD(HBOUND)
    THEN
        BEGIN {2}
          BLANKLINES(20,4,20);
          WRITELN('Low bound exceeds high bound error!');
          SQUAWK;
          WRITELN;
          STALL;
          EXIT(CHANGERANGE);
        END;  {2}

  ITEMINFO.HIGHANSWER := HBOUND;
  ITEMINFO.LOWANSWER := LBOUND;
  BLANKLINES(20,4,20);
  WRITELN('The new range is : ',ITEMINFO.LOWANSWER,'..',
          ITEMINFO.HIGHANSWER);
  WRITELN;
  STALL;
END;   (* change range *)

{--------------------------------------------------------------------}

(* change the answer  *)
PROCEDURE CHANGEANSWER;
VAR SELECT : CHAR;
    I : INTEGER;

    {...............................................................}

    (* get the new answer *)
    PROCEDURE GETANSWER;
    BEGIN
      SELECT := GETCHAR(['1'..'3'],TRUE,TRUE,TRUE);
      PAGE(OUTPUT);
      CASE SELECT OF
       '1' : BEGIN {1}
               GOTOXY(22,0);
               WRITE('MULTIPLE CHOICE/SINGLE ANSWER');
               ITEMINFO.ATYPE := CHARVALUE;
               GOTOXY(0,6);
               WRITE('Enter new answer and then press <RET> : ');
               FILLBUF(1,[ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER],
                       TRUE);
               ITEMINFO.ANSWER := LINEBUF[0];
               LINEBUF[0] := ' ';
             END;  {1}
       '2' : BEGIN {2}
               GOTOXY(22,0);
               WRITE('INTEGER VALUE ANSWER');
               ITEMINFO.ATYPE := INTVALUE;
               GOTOXY(0,6);
               WRITE('Enter new answer (integer ',
                     'value) and then press <RET> : ');
               READLN(I);
               ITEMINFO.INTANSWER := I;
             END;  {2}
       '3' : BEGIN {3}
               GOTOXY(22,0);
               WRITE('MULTIPLE CHOICE/MULTIPLE ANSWERS');
               ITEMINFO.ATYPE := SEVENCHR;
               GOTOXY(0,6);
               WRITE('Enter the number of answers',
                     ' this question has (1..7) : ');
               SELECT := GETCHAR(['1'..'7'],TRUE,TRUE,TRUE);
               ITEMINFO.ANSWERCOUNT := ORD(SELECT) - 48;
               WRITELN;
               WRITELN;
               FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
               BEGIN {4}
                 WRITE('Enter answer ',I,', then press <RET> : ');
                 FILLBUF(1,
                         [ITEMINFO.LOWANSWER..ITEMINFO.HIGHANSWER],
                         TRUE);
                 ITEMINFO.CHRANSWER[I] := LINEBUF[0];
                 LINEBUF[0] := ' ';
                 WRITELN;
               END;  {4}
             END;  {3}
      END;  (* case *)
    END;  (* get answer *)

{...................................................................}

BEGIN  (* change answer *)
  DECODEPRINT(MODBLOCK,MODPTR);
  GOTOXY(0,20);
  WRITE('Current answer(s) : ');
  CASE ITEMINFO.ATYPE OF
    CHARVALUE : WRITELN(ITEMINFO.ANSWER);
    INTVALUE  : WRITELN(ITEMINFO.INTANSWER);
    SEVENCHR  : BEGIN {1}
                  FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                    WRITE(ITEMINFO.CHRANSWER[I],' ');
                  WRITELN;
                END;  {1}
    END;  (* cases *)
    WRITELN;
    STALL;
    TEXT80MODE;
    GOTOXY(20,0);
    WRITE('SELECT ANSWER TYPE');
    GOTOXY(0,4);
    WRITE
      ('Select one of the following options by entering its number.');
    GOTOXY(16,8);
    WRITE('1.  MULTIPLE CHOICE/SINGLE ANSWER');
    GOTOXY(16,9);
    WRITE('2.  INTEGER VALUE ANSWER');
    GOTOXY(16,10);
    WRITE('3.  MULTIPLE CHOICE/MULTIPLE ANSWERS');
    GOTOXY(16,14);
    WRITE('Enter Choice # : ');
    GETANSWER;
    WRITELN;
    WRITELN;
    WRITE('The new answer(s) is : ');
    CASE ITEMINFO.ATYPE OF
      CHARVALUE : WRITELN(ITEMINFO.ANSWER);
      INTVALUE  : WRITELN(ITEMINFO.INTANSWER);
      SEVENCHR  : BEGIN {2}
                    FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                      WRITE(ITEMINFO.CHRANSWER[I],' ');
                      WRITELN;
                    END;  {2}
    END;  (* cases *)
    WRITELN;
    STALL;
END;  (* change answer *)

{-------------------------------------------------------------------}

(* reads item text file & displays page *)
PROCEDURE PAGEPRINT(VAR LASTPAGE : BOOLEAN);
VAR X,
    Y,
    SCREENBYTES,
    CHARCODE,
          SKIPBYTE : INTEGER;

          {..................................................................}

          (* returns next  code in item  file *)
          FUNCTION BUFFERCODE : INTEGER;
          BEGIN
            BUFFERCODE := TRIX.ASCIIBUF[MODPTR];
            MODPTR := MODPTR + 1;
            IF MODPTR > 2047
              THEN
                  BEGIN {1}
                    MODBLOCK := MODBLOCK + 4;
                    READITEMBLOCK(MODBLOCK);
                    MODPTR := 0;
                  END;  {1}
          END;  (* buffercode *)

      {......................................................................}

      BEGIN  (* page print *)
       SCREENBYTES :=
         (XSCREEN + 1) * (YSCREEN + 1);
       FILLCHAR(SCREEN[0],SCREENBYTES,' ');
       SETSCREEN;
       READITEMBLOCK(MODBLOCK);
       REPEAT
         CHARCODE := BUFFERCODE;
         CASE CHARCODE OF
           GOTOFLAG : BEGIN {1}
                        X := BUFFERCODE;
                        Y := BUFFERCODE;
                        SKIPBYTE := BUFFERCODE;
                        GOTOXY(X,Y);
                      END;  {1}
           PAGEFLAG : ;
           ENDITEM  : ;
         END;

         IF (CHARCODE >= 32) AND (CHARCODE <= 126)
           THEN
               BEGIN {2}
                 SCREEN[X,Y] := CHR(CHARCODE);
                 X := X + 1;
                 WRITE(CHR(CHARCODE));
               END;  {2}
       UNTIL (CHARCODE = ENDITEM) OR (CHARCODE = PAGEFLAG);

       IF CHARCODE = ENDITEM
         THEN
             LASTPAGE := TRUE
         ELSE
             LASTPAGE := FALSE;
     END;  (* page print *)

{............................................................................}

BEGIN  (* modify samples *)
  PAGE(OUTPUT);
  WRITELN;
  WRITELN('View what sample question?   (1..5)');
  WRITE('Enter the number then press <RET> : ');
  READLN(Z);
  ERROR := FALSE;

  IF (Z > MAXSAMPLES) OR (Z < 1)
    THEN
        ERROR := TRUE
    ELSE
        IF DIRECTORY.ITEMCODE[Z] < 0 THEN ERROR := TRUE;

  IF ERROR
    THEN
```

```
        BEGIN {1}
          WRITELN;
          WRITELN;
          WRITELN('No such sample question!');
          SQUAWK;
          WRITELN;
          STALL;
          EXIT(MODIFYSAMPLES);
        END;  {1}

  HASH_LOC := HASH(Z);
  RESET(FILEITEMINFO,DATANAME);
  SEEK(FILEITEMINFO,HASH_LOC);
  GET(FILEITEMINFO);
  ITEMINFO := FILEITEMINFO^;
  CLOSE(FILEITEMINFO,LOCK);

  REPEAT
    TEXT80MODE;
    GOTOXY(20,0);
    WRITE('MODIFY SAMPLE QUESTION MENU');
    GOTOXY(0,4);
    WRITE('Select one of the following options by entering its number.');
    GOTOXY(16,8);
    WRITE('1.  QUIT');
    GOTOXY(16,9);
    WRITE('2.  MODIFY SAMPLE TEXT');
    GOTOXY(16,10);
    WRITE('3.  MODIFY SAMPLE ANSWER');
    GOTOXY(16,11);
    WRITE('4.  MODIFY SAMPLE ANSWER RANGE');
    GOTOXY(16,12);
    WRITE('5.  MODIFY GRAPHICS FLAG');
    GOTOXY(16,16);
    WRITE('Enter Choice # : ');
    MODBLOCK := ITEMINFO.ITEMBLOCK;
    MODPTR := ITEMINFO.ITEMPTR;
    SELECT := GETCHAR(['1'..'5'],TRUE,TRUE,TRUE);
    CASE SELECT OF
      '1' : ;
      '2' : BEGIN {2}
              LOADPTRS;
              (* save ptrs to beginning of text *)
              ITEMINFO.ITEMBLOCK := CURRBLOCK;
              ITEMINFO.ITEMPTR := CURRFREEPTR;
              BLOCK := CURRBLOCK;
              BLOCKPTR := CURRFREEPTR;

              REPEAT
                PAGEPRINT(ENDTEXT);
                FILLSCREENBUFFER(TOPMAX,RIGHTMAX,BOTTOMMAX,LEFTMAX,FALSE);
                IF NOT (ENDTEXT)
                  THEN
                      CODESCREEN(FALSE);
              UNTIL ENDTEXT;

              CODESCREEN(TRUE);
              SAVEPTRS;
            END;  {2}
      '3' : CHANGEANSWER;
      '4' : CHANGERANGE;
      '5' : SETGFLAG;
    END;  (* cases *)
  UNTIL SELECT = '1';
  UPDATEITEMFILE(HASH_LOC);
END;  (* modifyitem *)

{---------------------------------------------------------------------------}

(* look at sample question *)
PROCEDURE VIEWSAMPLES;
VAR I,
    Z,
        BLOCK,
        BLOCKPTR : INTEGER;
        ERROR : BOOLEAN;

    BEGIN  (* view samples  *)
      PAGE(OUTPUT);
      WRITELN;
      WRITELN('View what sample question?   (1..5)');
      WRITE('Enter the number then press <RET> : ');
      READLN(Z);
      ERROR := FALSE;

      IF (Z > MAXSAMPLES) OR (Z < 1)
        THEN
            ERROR := TRUE
        ELSE
            IF DIRECTORY.ITEMCODE[Z] < 0 THEN ERROR := TRUE;

      IF ERROR
        THEN
            BEGIN {1}
              WRITELN;
              WRITELN('No such sample question!');
              SQUAWK;
              WRITELN;
              STALL;
              EXIT(VIEWSAMPLES);
            END;  {1}

      HASH_LOC := HASH(Z);
      RESET(FILEITEMINFO,DATANAME);
      SEEK(FILEITEMINFO,HASH_LOC);
      GET(FILEITEMINFO);
      ITEMINFO := FILEITEMINFO^;
      CLOSE(FILEITEMINFO,LOCK);
      BLOCK := ITEMINFO.ITEMBLOCK;
      BLOCKPTR := ITEMINFO.ITEMPTR;
      DECODEPRINT(BLOCK,BLOCKPTR);
      GOTOXY(0,20);
      WRITE('Answer(s) : ');
      CASE ITEMINFO.ATYPE OF
        CHARVALUE  : WRITELN(ITEMINFO.ANSWER);
        INTVALUE   : WRITELN(ITEMINFO.INTANSWER);
        SEVENCHR   : BEGIN {2}
                       FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                         WRITE(ITEMINFO.CHRANSWER[I],' ');
                       WRITELN;
                     END;  {2}
      END;  (* cases *)
      WRITELN;
      STALL;
    END;  (* view samples *)

{.............................................................................}

BEGIN  (* sample questions *)

  (* figure out how many sample questions there are *)
  SCOUNT := 0;
  FOR I := 1 TO MAXSAMPLES DO
    IF DIRECTORY.ITEMCODE[I] > 0 THEN SCOUNT := SCOUNT + 1;

  REPEAT
    MENU;
    COMMAND := GETCHAR(['1'..'5'],TRUE,FALSE,TRUE);
    CASE COMMAND OF
     '1' : ;
     '2' : ENTER_SAMPLES;
     '3' : VIEWSAMPLES;
     '4' : MODIFY_SAMPLES;
     '5' : DELETESAMPLE;
    END;
  UNTIL COMMAND = '1';
```

```
END;  (* sample questions *)
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : TMGR.DIR/T.LIST.TEXT         Volume : TFILES               *)
(*      Codefile : T.MGR.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(*               DEC.  1, 1982                  NPROC                         *)
(******************************************************************************)

(* lists the directory test names to the screen *)
SEGMENT PROCEDURE LISTTESTS(SHOWFILEINFO : BOOLEAN);
VAR I,J,K,ITEMCOUNT : INTEGER;
BEGIN
  PAGE(OUTPUT);
  GOTOXY(24,0);
  WRITE('LIST OF SUBTESTS');
  I := 0;
  J := 0;
  GOTOXY(0,3);

  REPEAT
    IF NOT (DIRINFO[I].NOTUSED)
      THEN
          BEGIN {1}
            J := J + 1;
            IF J <= 10
              THEN
                  GOTOXY(0,2+J)
              ELSE
                  GOTOXY(40,2+J-10);
            WRITE(J,'.  ',DIRINFO[I].TNAME);
            WRITELN(' (',DIRINFO[I].ITEMCOUNT,')');
          END;  {1}
    I := I + 1;
  UNTIL I > MAXSUBTESTS;

  IF SHOWFILEINFO
    THEN
        BEGIN {2}
          GOTOXY(0,18);
          WRITELN(J, ' files in directory, ', (MAXSUBTESTS - J + 1), ' unused');
          WRITELN;
          STALL;
        END;  {2}
END;  (* list tests *)

{----------------------------------------------------------------------------}

(* lists the tests in directory & loads *)
SEGMENT PROCEDURE LOADTEST(MESSAGE : STRING);
VAR Q,
    TESTNUM,
    RECNUM : INTEGER;
    OKTEST : BOOLEAN;
    TEXTCODE : CHAR;

BEGIN  (* load test *)
  OKTEST := FALSE;
  LISTTESTS(FALSE);
  RESET(FILEDIRECTORY,INDEXNAME);

  REPEAT
    GOTOXY(0,15);
    WRITELN('INSTRUCTIONS : Enter choice #, then press <RET>.');
    WRITE('                To escape, press 0 then <RET>.');
    GOTOXY(0,18);
    WRITE(MESSAGE);
   (*$I-*)
    READLN(TESTNUM);
   (*$I+*)

    IF TESTNUM = 0
      THEN
```

```
          BEGIN {1}
            ESCPROC := TRUE;
            CLOSE(FILEDIRECTORY,LOCK);
            EXIT(LOADTEST);
          END;  {1}

    IF (TESTNUM < 0) OR (TESTNUM >  (MAXSUBTESTS+1))
      THEN
          BEGIN {2}
            WRITELN;
            WRITELN('Invalid test # : ',TESTNUM);
            SQUAWK;
            WRITELN;
            STALL;
          END   {2}
      ELSE
          BEGIN {3}
            RECNUM := 0;
            Q := 0;

            REPEAT
              SEEK(FILEDIRECTORY,RECNUM);
              GET(FILEDIRECTORY);
              IF NOT (FILEDIRECTORY^.UNUSED) THEN Q := Q + 1;
              RECNUM := RECNUM + 1;
            UNTIL (Q = TESTNUM) OR (RECNUM > MAXSUBTESTS);

            IF Q = TESTNUM
              THEN
                  BEGIN {4}
                    CURRINDEXRECNUM := RECNUM - 1;
                    OKTEST := TRUE;
                  END   {4}
              ELSE
                  BEGIN {5}
                    WRITELN;
                    WRITELN('No test loaded');
                    WRITELN;
                    STALL;
                  END;  {5}
          END;  {3}
    IF NOT OKTEST THEN BLANKLINES(18,6,18);
  UNTIL OKTEST;

  SEEK(FILEDIRECTORY,CURRINDEXRECNUM);
  GET(FILEDIRECTORY);
  DIRECTORY := FILEDIRECTORY^;
  RIGHTMAX := 38;
  SCR80 := FALSE;
  VNORMAL := FALSE;
  TEXTCODE  := DIRECTORY.TESTNAME[1];

  CASE TEXTCODE OF
    '*' : BEGIN {6}
            RIGHTMAX := 78;
            SCR80 := TRUE;
            VNORMAL := FALSE;
          END;  {6}
    '@' : BEGIN {7}
            RIGHTMAX := 78;
            SCR80 := TRUE;
            VNORMAL := TRUE;
          END;  {7}
    '?' : BEGIN {8}
            RIGHTMAX := 38;
            SCR80 := FALSE;
            VNORMAL := TRUE;
          END;  {8}
  END;  (* cases *)
  CLOSE(FILEDIRECTORY,LOCK);
END;  (* load test *)
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : TMGR.DIR/T.NEW.TEXT          Volume : TFILES               *)
(*      Codefile : T.MGR.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(* File last modified : Jan 26, 1982                             NPRDC        *)
(******************************************************************************)

(* create a new test and save on disk *)
SEGMENT PROCEDURE NEWTEST;
VAR TEXTCODE,
    COMMAND : CHAR;

    {..........................................................................}

    (* returns ptr to unused directory space *)
    (* nil if none available *)

    FUNCTION DIRFREESLOT : INTEGER;
    VAR FOUND : BOOLEAN;
        SLOT : INTEGER;

    BEGIN  (* dir free slot *)
      RESET(FILEDIRECTORY,INDEXNAME);
      FOUND := FALSE;
      SLOT := 0;

      REPEAT
        SEEK(FILEDIRECTORY,SLOT);
        GET(FILEDIRECTORY);
        IF FILEDIRECTORY^.UNUSED
          THEN
              BEGIN {1}
                FOUND := TRUE;
                DIRECTORY := FILEDIRECTORY^;
              END   {1}
          ELSE
              SLOT := SLOT + 1;
      UNTIL (FOUND) OR (SLOT > MAXSUBTESTS);

      IF FOUND
        THEN
            DIRFREESLOT := SLOT
        ELSE
            DIRFREESLOT := NIL;
      CLOSE(FILEDIRECTORY,LOCK);
    END;  (* dir free slot *)

{..............................................................................}


    (* get the test name *)
    PROCEDURE GETTNAME;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(18,0);
      WRITE('SUBTEST NAME FORMAT INSTRUCTIONS');
      GOTOXY(0,2);
      WRITELN(
    'The first character of the subtest name specifies which screen format the ');
      WRITELN(
    'question text will appear in.  The screen codes are listed below.  The ');
      WRITELN(
    'second character of the subtest name specifies the total permitted time ');
      WRITELN(
    'in minutes for timed subtests.  The third and fourth characters specify');
      WRITELN(
    'the additional seconds permitted for timed subtests.  Type in the subtest');
      WRITFLN(
    'name then press <RET>.');
      WRITELN;
      WRITELN('''*'' as first char - inverse + 80 columns');
      WRITELN('''@'' as first char = normal + 80 columns');
      WRITELN('''?'' as first char = normal + 40 columns');
      WRITELN;
      WRITELN(
    'If first character of test name is not any of the above control characters,');
      WRITELN(
    'then the text format will default to inverse + 40 columns.');
      WRITELN;
      WRITELN;
      WRITELN;
      WRITELN('Enter the subtest name, then press <RET>');
      WRITELN;
      WRITE('----> ');
      READLN(DIRECTORY.TESTNAME);
    END;    (* gettname *)


BEGIN  (* newtest *)
  CURRINDEXRECNUM := DIRFREESLOT;
  IF CURRINDEXRECNUM = NIL
    THEN
        BEGIN {1}
          PAGE(OUTPUT);
          GOTOXY(0,5);
          WRITELN('No room in directory !!');
          WRITELN;
          STALL;
          EXIT(NEWTEST);
        END;  {1}
  DIRECTORY.UNUSED := FALSE;

  (* give instructions *)
  GETTNAME;

  RIGHTMAX := 38;
  SCR80 := FALSE;
  VNORMAL := FALSE;
  TEXTCODE  := DIRECTORY.TESTNAME[1];
  CASE TEXTCODE OF
    '%' : BEGIN {2}
            RIGHTMAX := 78;
            SCR80 := TRUE;
            VNORMAL := FALSE;
          END;  {2}
    '@' : BEGIN {3}
            RIGHTMAX := 78;
            SCR80 := TRUE;
            VNORMAL := TRUE;
          END;  {3}
    '?' : BEGIN {4}
            RIGHTMAX := 38;
            SCR80 := FALSE;
            VNORMAL := TRUE;
          END;  {4}
  END;  (* cases *)

  WITH DIRINFO[CURRINDEXRECNUM] DO
  BEGIN {5}
    NOTUSED := FALSE;
    TNAME := DIRECTORY.TESTNAME;
    ITEMCOUNT := 0;
  END;  {5}
  UPDATEDIRECTORY(CURRINDEXRECNUM);
  FETCHTEST;
END;  (* newtest *)
```

```
(*************************************************************************)
(*                                                                       *)
(*      Textfile : TMGR.DIR/T.DELETE.TEXT       Volume : TFILES          *)
(*      Codefile : T.MGR.CODE ('Include' file)  Volume : CATDATA         *)
(*                                                                       *)
(*************************************************************************)
(*                 DEC.  1, 1982                NPROC                    *)
(*************************************************************************)

SEGMENT PROCEDURE DELETETEST;
VAR I,
    DATASLOT,
    ITEMCNT : INTEGER;
    CHR : CHAR;

BEGIN  (* delete test *)
  PAGE(OUTPUT);
  GOTOXY(15,0);
  WRITELN('**********  WARNING  **********');
  WRITELN;
  WRITELN;
  WRITELN(
'You have selected the ''Delete Subtest'' option. This will purge the subtest');
  WRITELN(
'directory from the database and the itembanks for the subtest will cease to');
  WRITELN(
'exist');
  WRITELN;
  WRITELN;
  WRITE('Do you wish to continue?  Press ''N'' or ''Y'' : ');

  IF GETCHAR(['Y','N','y','n'],TRUE,TRUE,TRUE) IN ['N','n']
    THEN
        EXIT(DELETETEST);
  ITEMCNT := 0;
  LOADTEST('Delete which test? : ');
  IF ESCPROC THEN EXIT(DELETETEST);
  PAGE(OUTPUT);
  WRITE('Purge ',directory.testname,' ?  Press ''N'' or ''Y'' : ');

  IF GETCHAR(['Y','N','y','n'],TRUE,TRUE,TRUE) IN ['N','n']
    THEN
        EXIT(DELETETEST);
  WRITELN;
  WRITELN;
  WRITE('Deleting');
  FOR I := 0 TO MAXITEMPOOL DO
  BEGIN {1}
    WRITE('.');
    IF DIRECTORY.ITEMCODE[I] >= 0
      THEN
          BEGIN {2}
            ITEMCNT := ITEMCNT + 1;
            DIRECTORY.ITEMCODE[I] := NIL;
          END;  {2}
  END;  {1}
  DIRECTORY.UNUSED := TRUE;
  UPDATEDIRECTORY(CURRINDEXRECNUM);
  DIRINFO[CURRINDEXRECNUM].NOTUSED := TRUE;
END; (* delete test *)
```

```
(*****************************************************************************)
(*                                                                           *)
(*      Textfile : TMGR.DIR/T.FLOPPY.TEXT          Volume : TFILES           *)
(*      Codefile : T.MGR.CODE ('Include' file)     Volume : CATDATA          *)
(*                                                                           *)
(*****************************************************************************)
(* File last modified : Feb 10, 1983               NPRDC                     *)
(*****************************************************************************)

(* This procedure transfers a test directory, data, and text from the 10     *)
(* megabyte Corvus hard disc to a 5.5 inch soft floppy disc.  This procedure *)
(* is needed because the files on the Corvus are too large to fit on a floppy.*)
(* Thus we break up the three files of the test question data base into three *)
(* smaller files containing information for a single test. *)

SEGMENT PROCEDURE TRANSFERTO5INCH;
VAR I,
    J,
    DATARECNUM,
    FCURRBLOCK,                   (* current free block *)
    FCURRFREEPTR : INTEGER;       (* current free byte in free block *)

    FILENAME,                     (* prefix for filenames *)
    FDIRNAME,                     (* directory name *)
    FDATANAME,                    (* name of file for data *)
    FTEXTNAME      : STRING;      (* name of file for text *)

    (* directory *)
    FDIR           : DIRDATA;
    FDIRFILE       : FILE OF DIRDATA;

    (* question data *)
    FDATA          : ITEMDATA;
    FDATAFILE      : FILE OF ITEMDATA;

    (* question text *)
    FTEXT          : FILE;

    (* block sized buffer for ascii & control codes *)
    FITEMBUF       : PACKED ARRAY[0..2047] OF 0..139;

    {.........................................................................}

    (* This procedure gets a name for the files to be put on the floppy and   *)
    (* formats the files for the appropriate data. *)

    PROCEDURE FORMATFLOPPY;
    VAR NAMEOK : BOOLEAN;
        ERRNUM : INTEGER;

    BEGIN  (* format floppy *)
      LOADTEST('Write which subtest to floppy? : ');
      IF ESCPROC THEN EXIT(TRANSFERTO5INCH);
      NAMEOK := FALSE;
      PAGE(OUTPUT);

      (* get a filename for the test *)
      REPEAT
        WRITELN('Enter destination filename, then press <RET>');
        WRITELN;
        WRITE('---> ');
        READLN(FILENAME);
        IF FILENAME[1] = CHR(ESC) THEN EXIT(TRANSFERTO5INCH);
        FDIRNAME := CONCAT(FILENAME,'.DIR.DATA');
      (*$I-*)
        RESET(FDIRFILE,FDIRNAME);
      (*$I+*)

        IF IORESULT = 0
          THEN
              BEGIN {1}
                WRITE('Destroy old ',FDIRNAME,' ?  Press ''N'' or ''Y'' : ');
```

```
            IF GETCHAR(['Y','y','N','n'],TRUE,TRUE,TRUE) IN ['Y','y']
              THEN
                  BEGIN {2}
                    CLOSE(FDIRFILE,PURGE);
                    REWRITE(FDIRFILE,FDIRNAME);
                    NAMEOK := TRUE;
                  END;  {2}
            WRITELN;
          END   {1}
      ELSE
          BEGIN {3}
          (*$I-*)
            REWRITE(FDIRFILE,FDIRNAME);
          (*$I+*)
            ERRNUM := IORESULT;
            IF IORESULT <> 0
              THEN
                  BEGIN {4}
                    WRITELN;
                    WRITELN('Cannot open ',FDIRNAME,' IO error #',ERRNUM);
                    SQUAWK;
                    WRITELN;
                    WRITELN;
                  END   {4}
              ELSE
                  NAMEOK := TRUE
          END;  {3}
  UNTIL NAMEOK;

  (* transfer the directory *)
  SEEK(FDIRFILE,0);
  FDIR := DIRECTORY;
  FDIRFILE^ := FDIR;
  PUT(FDIRFILE);
  CLOSE(FDIRFILE,LOCK);

  (* make and format the data and ascii file for the test *)
  FDATANAME := CONCAT(FILENAME,'.DATA.DATA');
  FTEXTNAME := CONCAT(FILENAME,'.TEXT.DATA');
  REWRITE(FDATAFILE,FDATANAME);
  SEEK(FDATAFILE,MAXITEMPOOL);
  CLOSE(FDATAFILE,LOCK);
  REWRITE(FTEXT,FTEXTNAME);
  ERRNUM := BLOCKWRITE(FTEXT,FITEMBUF,4,0);
  CLOSE(FTEXT,LOCK);
END;  (* format floppy *)

{-------------------------------------------------------------------------}

(* This procedure writes a block of question ascii from the block sized   *)
(* buffer to the disk in the ascii file. *)

PROCEDURE FWRITEITEMBLOCK(WHICHBLOCK : INTEGER);
VAR BLOCKSTRANSFERRED,
    ERRNUM : INTEGER;
    BADIO : BOOLEAN;

BEGIN  (* fwrite item block *)
  BADIO := FALSE;
  RESET(FTEXT,FTEXTNAME);
  BLOCKSTRANSFERRED := BLOCKWRITE(FTEXT, FITEMBUF, 4, WHICHBLOCK);
  BADIO := ((BLOCKSTRANSFERRED < 4) OR (IORESULT <> 0));
  ERRNUM := IORESULT;
  CLOSE(FTEXT,LOCK);

  IF BADIO
    THEN
        BEGIN {1}
          WRITELN;WRITELN;
          WRITE('Block write io error # ',ERRNUM);
          WRITELN;
          STALL;
          EXIT(PROGRAM);
        END;  (1)
END;  (* fwriteitemblock *)

 {--------------------------------------------------------------------------}

 (* reads a block from disk into the item ascii buffer *)

PROCEDURE FREADITEMBLOCK(WHICHBLOCK : INTEGER);
VAR BLOCKSTRANSFERRED,
    ERRNUM : INTEGER;
    BADIO : BOOLEAN;

BEGIN  (* fread item block *)
  BADIO := FALSE;
  RESET(FTEXT,FTEXTNAME);
  BLOCKSTRANSFERRED := BLOCKREAD(FTEXT, FITEMBUF, 4, WHICHBLOCK);
  BADIO := ((BLOCKSTRANSFERRED < 4) OR (IORESULT <> 0));
  ERRNUM := IORESULT;
  CLOSE(FTEXT,LOCK);

  IF BADIO
    THEN
        BEGIN (1)
          WRITELN;WRITELN;
          WRITE('Block read io error # ',ERRNUM);
          WRITELN;
          STALL;
          EXIT(PROGRAM);
        END;  (1)
END;  (* fread item block *)

 {--------------------------------------------------------------------------}

 (* saves value of free space, block & byte ptr in block 0, bytes 0..3 of  *)
 (* text file.  *)

PROCEDURE FSAVEPTRS;
VAR TRIX : RECORD CASE INTEGER OF
             1 : (TWOBYTES : PACKED ARRAY
                               [0..1] OF CHAR);
             2 : (INTVALUE : INTEGER);
           END;

BEGIN  (* fsave ptrs *)
  FREADITEMBLOCK(0);
  TRIX.INTVALUE := FCURRBLOCK;
  MOVELEFT(TRIX.TWOBYTES[0],FITEMBUF[0],2);
  TRIX.INTVALUE := FCURRFREEPTR;
  MOVELEFT(TRIX.TWOBYTES[0],FITEMBUF[2],2);
  FWRITEITEMBLOCK(0);
END;   (* fsave ptrs *)

 {--------------------------------------------------------------------------}

 (* This procedure reads in the question text from the ascii file of the   *)
 (* corvus and block reads it onto the floppy.  *)

PROCEDURE ASCIITOFLOPPY(CBLOCK,CPTR : INTEGER);
VAR CORVUSBLOCK,
    CORVUSPTR    : INTEGER;

BEGIN  (* ascii to floppy *)
  CORVUSBLOCK := CBLOCK;
  CORVUSPTR := CPTR;
  READITEMBLOCK(CORVUSBLOCK);      (* get block where text starts *)
  FREADITEMBLOCK(FCURRBLOCK);      (* set buffer to fill *)

   (* read from corvus buffer into floppy buffer, if ptrs reach end *)
   (* of buffer, read in new block/write out full buffer             *)
  WHILE TRIX.ASCIIBUF[CORVUSPTR] <> ENDITEM DO
  BEGIN (1)
    FITEMBUF[FCURRFREEPTR] := TRIX.ASCIIBUF[CORVUSPTR];
    FCURRFREEPTR := FCURRFREEPTR + 1;
```

```
        CORVUSPTR := CORVUSPTR + 1;

        IF CORVUSPTR > 2047
          THEN
              BEGIN {2}
                CORVUSBLOCK := CORVUSBLOCK + 4;
                CORVUSPTR := 0;
                READITEMBLOCK(CORVUSBLOCK);
              END;  {2}

        IF FCURRFREEPTR > 2047
          THEN
              BEGIN {3}
                FWRITEITEMBLOCK(FCURRBLOCK);
                FCURRBLOCK := FCURRBLOCK + 4;
                FCURRFREEPTR := 0;
              END;  {3}
      END;  {1}

      FITEMBUF[FCURRFREEPTR] := ENDITEM;    (* mark end of text *)
      FCURRFREEPTR := FCURRFREEPTR + 1;
      FWRITEITEMBLOCK(FCURRBLOCK);

      IF FCURRFREEPTR > 2047
        THEN
            BEGIN {4}
              FCURRBLOCK := FCURRBLOCK + 4;
              FCURRFREEPTR := 0;
              FWRITEITEMBLOCK(FCURRBLOCK);
            END;  {4}
    END;   (* ascii to floppy *)

{..........................................................................}

BEGIN    (* transferto5inch *)
  FORMATFLOPPY;
  FCURRBLOCK := 0;
  FCURRFREEPTR := 4;    (* first four bytes reserved 0 - 3 *)
  RESET(FILEITEMINFO,DATANAME);
  RESET(FDATAFILE,FDATANAME);

  (* transfer data, text and update text pointers *)
  PAGE(OUTPUT);
  WRITELN('Transferring');
  J := 1;
  FOR I := 0 TO MAXITEMPOOL DO
  BEGIN {1}
    IF DIRECTORY.ITEMCODE[I] >= 0  (* question exists *)
      THEN
          BEGIN {2}
            WRITE('.');
            IF (J MOD 50) = 0 THEN WRITELN;
            J := J + 1;
            DATARECNUM := HASH(I);
            SEEK(FILEITEMINFO,DATARECNUM);
            GET(FILEITEMINFO);
            ITEMINFO := FILEITEMINFO^;
            FDATA := ITEMINFO;          (* transfer the data *)
            FDATA.ITEMBLOCK := FCURRBLOCK;   (* set new text ptrs *)
            FDATA.ITEMPTR := FCURRFREEPTR;
            SEEK(FDATAFILE,I);
            FDATAFILE^ := FDATA;
            PUT(FDATAFILE);                  (* write data to floppy *)

            (* transfer the text *)
            ASCIITOFLOPPY(ITEMINFO.ITEMBLOCK,ITEMINFO.ITEMPTR);
          END;  {2}
  END;  {1}
  FSAVEPTRS;                  (* save end of file marker *)
  CLOSE(FILEITEMINFO,LOCK);
  CLOSE(FDATAFILE,LOCK);
END;   (* transferto5inch *)
```

```
(***************************************************************************)
(*                                                                         *)
(*     Textfile : TMGR.DIR/T.SEARCH.TEXT       Volume : TFILES             *)
(*     Codefile : T.MGR.CODE ('Include' file)  Volume : CATDATA            *)
(*                                                                         *)
(***************************************************************************)
(*  File last modified : Feb 18, 1983          NPROC                       *)
(***************************************************************************)

(* list item text and/or item parameters *)
SEGMENT PROCEDURE SEARCHTEXT;


VAR SLOTNUM,
    SCREENBYTES,
    BLK,
    BLKPTR,
    I,
    K,
    NUMKEYS,
    KEYCOUNT,
    DATASLOT : INTEGER;

    DONELIST,
    CONSOLE,
    CONTINUE,
    COMPLETE,
    LISTTEXT : BOOLEAN;

    SELECT,
    OPT,
    COMMAND,
    LCOMMAND : CHAR;

    KEYWORD : ARRAY[1..10] OF STRING;

    ALREADYFOUND : ARRAY[1..10] OF BOOLEAN;

    RESULTS : ARRAY[0..MAXITEMPOOL] OF INTEGER;

    TEXTLINE,
    KEYSTR : STRING;


    {......................................................................}
    (* reads the item text file & displays item text *)
    PROCEDURE SCANQUEST(BLOCKNUM,BLOCKPTR : INTEGER);
    CONST MAXBUFSIZE = 2047;
          BLOCKSOUT = 4;
    VAR X,
        Y,
        B,
        CURRPTR,
        CURRBLK,
        CHARCODE,
        CHARCNT : INTEGER;

        BADIO : BOOLEAN;


        {----------------------------------------------------------------------}

        (* return the next code in ascii file *)
        FUNCTION BUFCODE : INTEGER;
        BEGIN
          BUFCODE := TRIX.ASCIIBUF[CURRPTR];
          CURRPTR := CURRPTR + 1;
          IF CURRPTR > MAXBUFSIZE THEN
          (* end of block/get next block and reset byte ptr *)
            BEGIN {1}
          CURRBLK := CURRBLK + BLOCKSOUT;
          READITEMBLOCK(CURRBLK);
          CURRPTR := 0;
        END;  {1}
    END;  (* bufcode *)

  {...........................................................................}

  BEGIN  (* scanquest *)

    FOR B := 1 TO 10 DO
      ALREADYFOUND[B] := FALSE;

    READITEMBLOCK(BLOCKNUM);
    (* set block/byte ptrs *)
    CURRPTR := BLOCKPTR;
    CURRBLK := BLOCKNUM;
    FILLCHAR(LINEBUF[0],80,' ');
    (* read bytes from the buffer *)
    REPEAT

      (* get char from block buffer *)
      CHARCODE := bufcode;

      CASE CHARCODE OF
        GOTOFLAG : BEGIN {1}    (* move cursor *)
                     (* next two bytes after flag are x,y coord *)
                     X := BUFCODE;
                     Y := BUFCODE;
                     CHARCNT := BUFCODE;
                     IF (CURRPTR + CHARCNT - 1) > MAXBUFSIZE THEN
                       BEGIN {2}
                         B := (MAXBUFSIZE + 1) - CURRPTR;
                         MOVELEFT(TRIX.ASCIIBUF[CURRPTR],LINEBUF[X],B);
                         X := X + B;
                         B := CHARCNT - B;
                         CURRBLK := CURRBLK + BLOCKSOUT;
                         READITEMBLOCK(CURRBLK);
                         CURRPTR := 0;
                         MOVELEFT(TRIX.ASCIIBUF[CURRPTR],LINEBUF[X],B);
                         CURRPTR := CURRPTR + B;
                       END   {2}
                       ELSE
                       BEGIN {3}
                         MOVELEFT(TRIX.ASCIIBUF[CURRPTR],LINEBUF[X],CHARCNT);
                         CURRPTR := CURRPTR + CHARCNT;
                         IF CURRPTR > MAXBUFSIZE THEN
                           BEGIN {4}
                             CURRBLK :=  CURRBLK + BLOCKSOUT;
                             CURRPTR := 0;
                             READITEMBLOCK(CURRBLK);
                           END;  {4}
                       END;  {3}

                       MOVELEFT(LINEBUF[0],TEXTLINE[1],80);
                       FOR K := 1 TO NUMKEYS DO
                       BEGIN
                         KEYSTR := KEYWORD[K];
                         IF POS(KEYSTR,TEXTLINE) <> 0 THEN
                         BEGIN
                           IF NOT ALREADYFOUND[K] THEN
                           BEGIN
                             KEYCOUNT := KEYCOUNT + 1;
                             ALREADYFOUND[K] := TRUE;
                           END;
                         END;
                       END;
                     FILLCHAR(LINEBUF[0],80,' ');
                   END;  {1}
        PAGEFLAG : ;
        ENDITEM  : ;

    END;
```

```
   UNTIL CHARCODE = ENDITEM; (* until end flag hit *)
  END;  (* scanquest *)




   {-----------------------------------------------------------------------}

   (* reads item text file & displays item text to printer or file *)
   PROCEDURE LISTPRINT(BLOCKNUM, BLOCKPTR : INTEGER);
   VAR X,
       Y,
       OLDY,
       CURRPTR,
       CURRBLK,
       SCREENBYTES,
       CHARCODE,
       SKIPBYTE : INTEGER;

       {..................................................................}

       (* returns next code in file *)
       FUNCTION LBUFCODE : INTEGER;
       BEGIN
         LBUFCODE := TRIX.ASCIIBUF[CURRPTR];
         CURRPTR := CURRPTR + 1;
         IF CURRPTR > 2047 THEN
         BEGIN {1}
           CURRBLK := CURRBLK + 4;
           READITEMBLOCK(CURRBLK);
           CURRPTR := 0;
         END;  {1}
     END;  (* lbufcode *)

  {.........................................................................}

  BEGIN  (*  listprint *)
    SCREENBYTES := (XSCREEN + 1) * (YSCREEN + 1);
    FILLCHAR(SCREEN[0],SCREENBYTES,' ');
    OLDY := TOPMAX;
    READITEMBLOCK(BLOCKNUM);
    CURRPTR := BLOCKPTR;
    CURRBLK := BLOCKNUM;
    REPEAT
      CHARCODE := LBUFCODE;
      CASE CHARCODE OF
        GOTOFLAG : BEGIN {1}
                     X := LBUFCODE;
                     Y := LBUFCODE;
                     (* ignore next byte, due to a file modification *)
                     SKIPBYTE := LBUFCODE;
                     WHILE OLDY < Y DO
                     BEGIN {2}
                       WRITELN(DEST);
                       OLDY := OLDY + 1;
                     END;  {2}
                     WRITE(DEST,' ' : X);
                   END;  {1}
        PAGEFLAG : BEGIN {3}
                     WRITELN(DEST);
                     WRITELN(DEST);
                     WRITELN(DEST);
                     OLDY := TOPMAX;
                   END;  {3}
        ENDITEM  : ;
      END; (* cases *)
      IF (CHARCODE >= 32) AND (CHARCODE <= 126) THEN
      BEGIN {4}
        SCREEN[X,Y] := CHR(CHARCODE);
        X := X + 1;
        WRITE(DEST,CHR(CHARCODE));
      END;  {4}
  UNTIL CHARCODE = ENDITEM;
END;  (* print *)

{-------------------------------------------------------------------------}

(* lists things to the console *)
PROCEDURE LCONSOLE;
VAR FIXCHAR : CHAR;
    GRAF : BOOLEAN;
BEGIN
  DECODEPRINT(BLK,BLKPTR);
  GOTOXY(0,20);
  IF SLOTNUM > MAXSAMPLES THEN
  BEGIN {3}
    WRITE('Item code : ',DIRECTORY.ITEMCODE[SLOTNUM]);
    GOTOXY(0,21);
    WRITE('Answer :');
    CASE ITEMINFO.ATYPE OF
      CHARVALUE : WRITELN(' ',ITEMINFO.ANSWER);
      INTVALUE  : WRITELN(' ',ITEMINFO.INTANSWER);
      SEVENCHR  : BEGIN
                    FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                      WRITE(' ',ITEMINFO.CHRANSWER[I]);
                    WRITELN;
                  END;
    END;  (* cases *)
    GOTOXY(0,23);
    WRITE('Press <RET> to continue, <ESC> to quit ');
    IF GETCHAR([CHR(RET),CHR(ESC)],TRUE,TRUE,TRUE) = CHR(ESC) THEN
    BEGIN {5}
      CLOSE(FILEITEMINFO,LOCK);
      EXIT(SEARCHTEXT);
    END   {4}
  END;
END; (* lconsole *)

{-------------------------------------------------------------------------}

(* lists item text and data to file/printer *)
PROCEDURE LFILE;
BEGIN
  IF SLOTNUM > MAXSAMPLES THEN
  BEGIN {2}
    WRITELN(DEST,' Item code : ',DIRECTORY.ITEMCODE[SLOTNUM]);
    WRITELN(DEST);
    WRITELN(DEST,' A parameter : ',ITEMINFO.A);
    WRITELN(DEST,' B parameter : ',ITEMINFO.B);
    WRITELN(DEST,' C parameter : ',ITEMINFO.C);
    WRITELN(DEST);WRITELN(DEST);
  END;  {2}
  LISTPRINT(BLK,BLKPTR);
  WRITELN(DEST);
  WRITELN(DEST);
  IF SLOTNUM > MAXSAMPLES THEN
  BEGIN {3}
    WRITE(DEST,' Answer(s) : ');
    CASE ITEMINFO.ATYPE OF
      CHARVALUE : WRITELN(DEST,ITEMINFO.ANSWER);
      INTVALUE  : WRITELN(DEST,ITEMINFO.INTANSWER);
      SEVENCHR  : BEGIN {4}
                    FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                      WRITE(DEST,ITEMINFO.CHRANSWER[I],' ');
                    WRITELN(DEST);
                  END;  {4}
    END;  (* cases *)
    WRITELN(DEST);
  END;  {3}
  FOR I := 1 TO 80 DO
    WRITE(DEST,'-');
  WRITELN(DEST);WRITELN(DEST);
END;  (* lfile *)

{-------------------------------------------------------------------------}
```

```
    PROCEDURE GETSEARCHINFO;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(20,0);
      WRITE('ENTER KEYWORDS TO SEARCH FOR');
      WRITELN;
      REPEAT
        WRITELN;
        WRITELN(
'You may have up to 10 keywords.  How many keywords would you like to enter?');
        WRITELN;
        WRITE(
'Enter # of key words and then press <RET> : ');
        READLN(NUMKEYS);
        IF NUMKEYS > 10 THEN SQUAWK;
        IF NUMKEYS <= 0 THEN
          EXIT(SEARCHTEXT);
      UNTIL NUMKEYS <= 10;

      WRITELN;
      WRITELN;
      WRITELN(
'************         Enter your keyword and then press <RET>          ************');
      WRITELN;
      FOR I := 1 TO NUMKEYS DO
      BEGIN
        WRITELN;
        WRITE('Keyword ',I,' : ');
        READLN(KEYSTR);
        KEYWORD[I] := KEYSTR;
      END;



      PAGE(OUTPUT);
      GOTOXY(20,0);
      WRITE('OUTPUT SELECT MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following options by entering its number.');
      GOTOXY(16,8);
      WRITE('1.  QUIT');
      GOTOXY(16,9);
      WRITE('2.  SEARCH RESULTS TO CONSOLE');
      GOTOXY(16,10);
      WRITE('3.  SEARCH RESULTS TO PRINTER');
      GOTOXY(16,11);
      WRITE('4.  SEARCH RESULTS TO FILE');
      GOTOXY(16,18);
      WRITE('Enter Choice # : ');
      SELECT :=  GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
      CONSOLE := FALSE;

      CASE SELECT OF
        '1' : EXIT(SEARCHTEXT);
        '2' : CONSOLE := TRUE;
        '3' : REWRITE(DEST,UNITNUMPRINTER);
        '4' : GETNEWFILE;
      END;
    END;   (* get searchinfo *)

{.............................................................................}

BEGIN  (* searchtext  *)

  (* get the subtest you want to search *)
  LOADTEST('Search text in which subtest? : ');

  IF ESCPROC THEN EXIT(SEARCHTEXT);

  TEXTLINE :=
'                                                                                ';
  FOR I := 0 TO MAXITEMPOOL DO
```

```
    RESULTS[I] := 0;


  GETSEARCHINFO;
  PAGE(OUTPUT);
  WRITE('Searching');

  SLOTNUM := MAXSAMPLES + 1;
  RESET(FILEITEMINFO,DATANAME);

  REPEAT

    IF DIRECTORY.ITEMCODE[SLOTNUM] >= 0 THEN
    BEGIN {10}
      WRITE('.');
      DATASLOT := HASH(SLOTNUM);
      SEEK(FILEITEMINFO,DATASLOT);
      GET(FILEITEMINFO);
      ITEMINFO := FILEITEMINFO^;
      BLK := ITEMINFO.ITEMBLOCK;
      BLKPTR := ITEMINFO.ITEMPTR;


      IF NOT ITEMINFO.GRAPHICS THEN
      BEGIN
        KEYCOUNT := 0;
        SCANQUEST(BLK,BLKPTR);
        RESULTS[SLOTNUM] := KEYCOUNT;
      END;
    END;  {10}
    SLOTNUM := SLOTNUM + 1;
  UNTIL (SLOTNUM > MAXITEMPOOL);

  PAGE(OUTPUT);
  FOR K := 1 TO NUMKEYS DO
  BEGIN
    WRITELN;
    WRITE(
'**********          Questions which contained ',K,' key(s)          *********');
    WRITELN;
    WRITELN;
    FOR I := 0 TO MAXITEMPOOL DO
    BEGIN
      IF RESULTS[I] = K THEN
        WRITE(DIRECTORY.ITEMCODE[I],'       ');
    END;
    WRITELN;
    WRITELN;
    STALL;
    WRITELN;
    WRITELN;
  END;


  IF NOT (CONSOLE) THEN
    CLOSE(DEST,LOCK);
END;  (* listitems *)
```

EMGR.DIR:
Subdirectory - Examinee Manager Textfiles

```
(*$S+*)
(*****************************************************************************)
(*                                                                           *)
(*      Textfile : EMGR.DIR/E.MGR.TEXT          Volume : TFILES              *)
(*      Codefile : E.MGR.CODE                   Volume : CATDATA             *)
(*                                                                           *)
(*****************************************************************************)
(*                            VERSION 1.03                     NPRDC         *)
(*****************************************************************************)
(*  File last modified  :     Feb 28, 1983                                   *)
(*****************************************************************************)
(*                                                                           *)
(* Functions : 1) allows access to examinee records.                         *)
(*             2) lists examinee's test scores.                              *)
(*             3) removing, or modifying an examinee record.                 *)
(*             4) allows examinee access to system by specifying who can     *)
(*                enter the cat system.                                      *)
(*             5) lists the status of all examinees in the system.           *)
(*             6) allows data entry of examinee records                      *)
(*                                                                           *)
(*****************************************************************************)
(*                                                                           *)
(*                      Program Modification History                         *)
(*                                                                           *)
(*      DATE                                    CHANGE                       *)
(* ------------------------------------------------------------------------- *)
(*    1 / 4 / 83          Personal data record changed, files reformatted.   *)
(*                        New fields include branch of service, post ASVAB   *)
(*                        test scores, and subtest order. Designated soft-   *)
(*                        ware version changed from 1.01 to 1.02.            *)
(*                                                                           *)
(*    1 / 13 / 83         Data verification sequence appears after entry of  *)
(*                        personal data for 3 fields : 1. Last Name,         *)
(*                        2. ASVAB Scores, and 3. Rating/MOS.                *)
(*                                                                           *)
(*****************************************************************************)

PROGRAM PERSONMGR;
USES CHAINSTUFF;

CONST (* ascii values *)
      BELL = 7;
      NUL = 0;
      LARROW = 8;
      RARROW = 21;
      RET = 13;
      ESC = 27;
      SPACE = 32;
      ASCIIOFFSET = 48;   (* ascii zero *)

      NIL = -1;

      (* available space in directory *)
      MAXEXAMINEE = 50;

      ZINDEXNAME = 'CATDATA:EINDEX.DATA';
      ZINFONAME = 'CATDATA:EINFO.DATA';
      ZPRESULTS = 'CATDATA:ERESULTS.DATA';

      (* examinee personal data files *)
      ZPINFONAME = 'CATDATA:EPDATA.DATA';

      DONEINDEX = 'CATDATA:DONEDIR.DATA';
      DONEINFO = 'CATDATA:DONEINFO.DATA';
      DONERESULTS = 'CATDATA : DONERSLTS.DATA';
      DONEPINFO = 'CATDATA : DONEEPD.DATA';

      (* output file for test scores *)
      DEFAULTFILE = 'CATDATA:EXAMINEE.DATA';

      (* maximum # of questions per subtest *)
      QUESTIONS = 28;
      (* maximum # of subtests per examinee *)
      STESTS = 20;

      (* subtest directory *)
      TINDEXNAME = 'CATDATA:TESTINDEX.DATA';

      MAXITEMPOOL = 300;

      (* maximum allowable subtests given *)
      GMAXSUBTEST = 20;

      (* maximum spaces in directory *)
      MAXSUBTESTS = 20;

      (* printer unit number *)
      UNITNUMPRINTER = 'PRINTER:';

      VERSION = '[1.03]';


TYPE  (* test directory *)
     DIRDATA = PACKED RECORD
                      UNUSED      : BOOLEAN;
                      TESTNAME    : STRING;
                      ITEMCODE    : PACKED ARRAY
                                       [0..MAXITEMPOOL]
                                       OF INTEGER;
                    END;

      (* social security # *)
      IDTYPE = PACKED ARRAY[0..8] OF CHAR;

      (* coding speed answer *)
      SEVENTYPE =  PACKED ARRAY[1..7] OF CHAR;

      (* response types *)
      ITEMRESPONSES = (CHARVALUE,INTVALUE,SEVENCHR);

      SETOFCHAR = SET OF CHAR;

      (* examinee directory *)
      INDEX = PACKED ARRAY[0..MAXEXAMINEE] OF PACKED RECORD

                                             (* true ==> trecord available *)
                                             UNUSED : BOOLEAN;

                                             (* social security/id key *)
                                             ID : IDTYPE;
                                           END;

      (* examinee pre-testing data *)
      EXAMEINFO = PACKED RECORD

                   (* social security # *)
                   ID           : IDTYPE;

                   ORIENTATIONTIME,
                   PREVTIMELASTTEST,
                   (* number of proctor calls *)
                   NUMPROC,

                   (* total time spent at computer *)
                   TOTTIMECONSOLE,

                   (* number of key in errors *)
                   NUMERRORS,

                   (* # of last test taken - 1, eg.- if on test 5 then *)
                   (* this variable holds a value of 4.              *)
                   LASTTEST      : INTEGER;

                   (* date of log in *)
```

```
              DATE          : PACKED ARRAY[0..5] OF CHAR;

              (* time at log in *)
              TIME          : PACKED ARRAY[0..3] OF CHAR;

              (* testing configuration given to this examinee *)

              (* record # of subtest directories *)
              TESTORDER,

              (* adaptive strategy *)
              STRATEGY,

              (* # of questions per test *)
              TESTLENGTH    : PACKED ARRAY[1..GMAXSUBTEST]
                              OF 0..128;

              (* initial variance for each test *)
              CKERROR       : ARRAY[1..GMAXSUBTEST] OF REAL;

              (* flags to control flow of subtests *)
              SUBSTOP       : PACKED ARRAY[1..GMAXSUBTEST] OF BOOLEAN;

            END;

(* examinee personal data *)
PINFOREC = RECORD
             LASTNAME : PACKED ARRAY[0..14] OF CHAR;
             FIRSTNAME : PACKED ARRAY[0..11] OF CHAR;
             MINITIAL : CHAR;
             CURRADDRESS,
             HOMEOFREC  : PACKED ARRAY[0..1] OF CHAR;
             CITIZENSHIP : PACKED ARRAY[0..3] OF CHAR;
             SEX : CHAR;
             POPGROUP : PACKED ARRAY[0..4] OF CHAR;
             ETHNIC : PACKED ARRAY[0..1] OF CHAR;
             MARITAL : CHAR;
             DEPENDANTS : PACKED ARRAY[0..1] OF CHAR;
             BIRTHDATE : PACKED ARRAY[0..7] OF CHAR;
             EDUCATION : PACKED ARRAY[0..2] OF CHAR;
             TESTID : PACKED ARRAY[0..2] OF CHAR;
             AFQT : PACKED ARRAY[0..1] OF CHAR;
             ASVAB  : PACKED ARRAY[0..43] OF CHAR;
             ENLISTDATE,
             ACTSERDATE  : PACKED ARRAY[0..7] OF CHAR;
             ENL : PACKED ARRAY[0..4] OF CHAR;
             AFEES : PACKED ARRAY[0..1] OF CHAR;
             BOS : PACKED ARRAY[0..1] OF CHAR;
             POSTASVAB : PACKED ARRAY[0..43] OF CHAR;
             STESTORDER : PACKED ARRAY[0..59] OF CHAR;
           END;

(* question scores , data on examinee with respect to question *)
ITEM = PACKED RECORD

         ACORRECT : PACKED ARRAY[0..6] OF BOOLEAN;
         ACOUNT,

         (* which question he took, code # *)
         ITEMNUM  : INTEGER;

         (* true ==> answered it correctly *)
         CORRECT  : BOOLEAN;

         (* ability after answering this question *)
         THETA,

         (* variance after answering this question *)
         ERROR,

         (* time spent answering this question *)
         LATENCY  : REAL;
```

```
               (* how he answered the question *)
               CASE RTYPE : ITEMRESPONSES OF
                 CHARVALUE : (RESPONSE : CHAR);
                 INTVALUE  : (INTRESPONSE : INTEGER);
                 SEVENCHR  : (CHRRESPONSE : SEVENTYPE);
             END;

      (* test scores of examinee results  *)
      SUBTEST = PACKED RECORD

                  STTIME,
                  STINSTRTIME,
                  STPROCTCALLS,
                  MODSTTIME        : INTEGER;

                  (* number of questions he answered *)
                  NUMITEMS,

                  (* # of correct responses *)
                  NUMCORR  : 0..128;

                  (* final estimate of ability *)
                  ESTABILITY,

                  (* final variance of ability *)
                  VARIANCE     : REAL;

                  (* results on question level *)
                  ITEMINFO     : PACKED ARRAY[0..QUESTIONS] OF ITEM;
               END;

VAR (* file names *)
    PINFONAME,
    INDEXNAME,
    INFONAME,
    RESULTS    : STRING;

    BACKUP : BOOLEAN;

    LETTERS,DIGITS,CHARACTERS : SET OF CHAR;
    OUTPUT,
    COMMAND : CHAR;

    EXAMINEE : EXAMEINFO;
    FILEEXAMINEE : FILE OF EXAMEINFO;

    DIR : INDEX;
    FILEDIR : FILE OF INDEX;

    TESTS : SUBTEST;
    FILETESTS : FILE OF SUBTEST;

    (* test directory *)
    DIRECTORY : DIRDATA;
    FILEDIRECTORY : FILE OF DIRDATA;

    (* examinee personal data *)
    PINFO : PINFOREC;
    PINFOFILE : FILE OF PINFOREC;

    (* output file for test results *)
    DEST : TEXT;

    (* string buffer *)
    LINEBUF : PACKED ARRAY[0..79] OF CHAR;

PROCEDURE PAGE(DUMMY : CHAR); FORWARD;
PROCEDURE TOPOFFORM;  FORWARD;
PROCEDURE SQUAWK;  FORWARD;
PROCEDURE BLANKLINES(START,COUNT,ENDCURSOR : INTEGER); FORWARD;
```

```
FUNCTION GETCHAR(OKSET : SETOFCHAR;
                 FLUSHQUEUE,ECHO,BEEP : BOOLEAN) : CHAR;  FORWARD;
PROCEDURE STALL;  FORWARD;
PROCEDURE FILLBUF(CHARCNT : INTEGER;
                  OKSET : SETOFCHAR; ERASE : BOOLEAN);  FORWARD;
PROCEDURE  GETNEWFILE;  FORWARD;
PROCEDURE SCRCONTROL(I,J,K : INTEGER); FORWARD;
PROCEDURE TEXT40MODE;     FORWARD;
PROCEDURE TEXT80MODE;     FORWARD;
PROCEDURE INVERSE;        FORWARD;
PROCEDURE NORMAL;         FORWARD;
PROCEDURE LOADPDATA(RECORDNUM : INTEGER);    FORWARD;
PROCEDURE SAVEPDATA(RECORDNUM : INTEGER);    FORWARD;
PROCEDURE LOADINDEX;  FORWARD;
PROCEDURE LOADEXAMINEE(RECNUM : INTEGER);    FORWARD;
PROCEDURE UPDATEEXAMINEE(RECNUM : INTEGER); FORWARD;
PROCEDURE UPDATEINDEX;    FORWARD;
PROCEDURE LOADRESULTS(RECNUM : INTEGER);  FORWARD;
PROCEDURE UPDATERESULTS(RECNUM : INTEGER);    FORWARD;
PROCEDURE INITEREC;  FORWARD;
FUNCTION DIRINDEXNUM(IDNUM : IDTYPE) : INTEGER;    FORWARD;
PROCEDURE ENTERID;  FORWARD;

(* utility subroutines *)
   (*$I /TFILES/EMGR.DIR/E.UTL.TEXT *)

(* disk io routines *)
   (*$I /TFILES/EMGR.DIR/E.IO.TEXT *)

(* examinee manager subroutines *)
   (*$I /TFILES/EMGR.DIR/E.SUBRT.TEXT *)

(* log - in examinee routines *)
   (*$I /TFILES/EMGR.DIR/E.LOGIN.TEXT *)

(* list status of examinees in directory *)
   (*$I /TFILES/EMGR.DIR/E.STATUS.TEXT *)

(* list examinee file data *)
   (*$I /TFILES/EMGR.DIR/E.SUMMARY.TEXT *)

(* remove an examinee from directory *)
   (*$I /TFILES/EMGR.DIR/E.DELETE.TEXT *)

(* fetch examinee routines *)
   (*$I /TFILES/EMGR.DIR/E.FETCH.TEXT *)

(* set up files/directory *)
   (*$I /TFILES/EMGR.DIR/E.ZERO.TEXT *)

(* end of day routines *)
   (*$I /TFILES/EMGR.DIR/E.ENDOFDAY.TEXT *)

    {.............................................................................}

    PROCEDURE MENU;
    VAR X,Y : INTEGER;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(18,0);
      WRITE('EXAMINEE MANAGER MENU ',VERSION);
      GOTOXY(0,4);
      WRITE('Select one of the following procedures by entering its number.');
      X := 16;
      Y := 8;
      GOTOXY(X,Y);
      WRITE('1.  QUIT');
      GOTOXY(X,Y+1);
      WRITE('2.  LOG EXAMINEE INTO SYSTEM');
      GOTOXY(X,Y+2);
      WRITE('3.  ENTER EXAMINEE PERSONAL DATA');
      GOTOXY(X,Y+3);
      WRITE('4.  VIEW EXAMINEE RECORDS');
```

6SS

```
  GOTOXY(X,Y+4);
  WRITE('5.  LIST DIRECTORY');
  GOTOXY(X,Y+5);
  WRITE('6.  SEPARATE DONE/NOT DONE EXAMINEES');
  GOTOXY(X,Y+6);
  WRITE('7.  SELECT FILES TO MANAGE');
  GOTOXY(X,Y+7);
  WRITE('8.  REMOVE EXAMINEE');
  GOTOXY(X,Y+8);
  WRITE('9.  PURGE DIRECTORY');
  GOTOXY(0,6);
  WRITE('Currently managing ');
  INVERSE;
  IF BACKUP THEN
    WRITE('DONE EXAMINEE')
  ELSE
    WRITE('SESSION');
  NORMAL;
  WRITE(' files.');
  GOTOXY(X,Y+11);
  WRITE('Enter Choice # : ');
END;  (* menu *)

{---------------------------------------------------------------------------}

(* specify which files you want to look at *)
PROCEDURE FILESPECIFY;
VAR FCOMMAND : CHAR;
BEGIN
  PAGE(OUTPUT);
(*$I-*)
  RESET(FILEDIR,DONEINDEX);
(*$I+*)
  IF IORESULT <> 0 THEN
  BEGIN {1}
    WRITELN;
    WRITELN('Back-up files do not exist.');
    WRITELN;
    STALL;
    EXIT(FILESPECIFY);
  END   {1}
  ELSE
  BEGIN {2}
    CLOSE(FILEDIR,NORMAL);
    GOTOXY(20,0);
    WRITE('SPECIFY FILES MENU');
    GOTOXY(0,4);
    WRITE('Select one of the following options by entering its number.');
    GOTOXY(16,9);
    WRITE('1.  QUIT');
    GOTOXY(16,10);
    WRITE('2.  MANAGE SESSION FILES');
    GOTOXY(16,11);
    WRITE('3.  MANAGE DONE EXAMINEES');
    REPEAT
      GOTOXY(0,6);
      WRITE('You currently have option : ');
      IF BACKUP THEN
        WRITE('3')
      ELSE
        WRITE('2');
      INVERSE;
      IF BACKUP THEN
      BEGIN {3}
        GOTOXY(16,11);
        WRITE('3.  MANAGE DONE EXAMINEES');
      END   {3}
      ELSE
      BEGIN {4}
        GOTOXY(16,10);
        WRITE('2.  MANAGE SESSION FILES');
      END;  {4}
      NORMAL;
```

```
          GOTOXY(16,15);
          WRITE('                     ');
          GOTOXY(16,15);
          WRITE('Enter Choice # : ');
          FCOMMAND :=  GETCHAR(['1'..'3'],TRUE,TRUE,TRUE);
          IF BACKUP THEN
          BEGIN {5}
            GOTOXY(16,11);
            WRITE('3.  MANAGE DONE EXAMINEES');
          END   {5}
          ELSE
          BEGIN {6}
            GOTOXY(16,10);
            WRITE('2.  MANAGE SESSION FILES');
          END;  {6}
          CASE FCOMMAND OF
            '1' : EXIT(FILESPECIFY);
            '2' : BEGIN {7}
                    INDEXNAME := ZINDEXNAME;
                    INFONAME := ZINFONAME;
                    RESULTS := ZRESULTS;
                    PINFONAME := ZPINFONAME;
                    BACKUP := FALSE;
                  END;  {7}
            '3' : BEGIN {8}
                    INDEXNAME := DONEINDEX;
                    INFONAME := DONEINFO;
                    RESULTS := DONERESULTS;
                    PINFONAME := DONEPINFO;
                    BACKUP := TRUE;
                  END   {8}
          END;  (* cases *)
        UNTIL FCOMMAND = '1';
      END;  {2}
    END;   (* filespecify *)

{.............................................................................}

BEGIN  (* main program *)
  PAGE(OUTPUT);
  DIGITS := ['0'..'9'];
  LETTERS := ['A'..'Z','a'..'z'];
  CHARACTERS := [CHR(32)..CHR(126)];
  FILLCHAR(LINEBUF[0],80,' ');

  (* files required *)
  INDEXNAME := ZINDEXNAME;
  INFONAME := ZINFONAME;
  RESULTS := ZRESULTS;
  PINFONAME := ZPINFONAME;
  BACKUP := FALSE;

  REPEAT
    MENU;
    COMMAND := GETCHAR(['1'..'9'],TRUE,TRUE,TRUE);
    CASE COMMAND OF
      '1' : ;
      '2' : BEGIN {1}
              INDEXNAME := ZINDEXNAME;
              INFONAME := ZINFONAME;
              RESULTS := ZRESULTS;
              PINFONAME := ZPINFONAME;
              LOGINEXAMINEE;
              IF BACKUP THEN
              BEGIN {2}
                INDEXNAME := DONEINDEX;
                INFONAME := DONEINFO;
                RESULTS := DONERESULTS;
                PINFONAME := DONEPINFO;
              END;  {2}
            END;  {1}
      '3' : ENTERDATA;
      '4' : ESUMMARY;
      '5' : ESTATUS;
      '6' : BEGIN {3}
              ENDOFDAY;
              IF BACKUP THEN
              BEGIN {4}
                INDEXNAME := DONEINDEX;
                INFONAME := DONEINFO;
                RESULTS := DONERESULTS;
                PINFONAME := DONEPINFO;
              END;  {4}
            END;  {3}
      '7' : FILESPECIFY;
      '8' : DELETEEXAMINEE;
      '9' : ZERODIRECTORY;
    END;  (* cases *)
  UNTIL COMMAND = '1';
  PAGE(OUTPUT);
  GOTOXY(15,10);
  WRITE('Loading Catproject driver...');
  SETCHAIN('CATDATA:CATPROJECT');

END.
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : EMGR.DIR/E.UTL.TEXT          Volume : TFILES               *)
(*      Codefile : E.MGR.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(*               DEC.  1, 1982                  NPROC                         *)
(******************************************************************************)

(* clear the screen *)
PROCEDURE PAGE;
BEGIN

  (* for apple 2 *)
  WRITE(CHR(12));

  (* for apple 3 *)
  WRITE(CHR(28));
  GOTOXY(0,0);
END;  (* page *)

{------------------------------------------------------------------------------}

(* form feeds the printer *)
PROCEDURE TOPOFFORM;
BEGIN
  REWRITE(DEST,UNITNUMPRINTER);
  WRITE(DEST,CHR(12));
  CLOSE(DEST,NORMAL);
END;   (* top of form *)

{------------------------------------------------------------------------------}

(**** rings the bell ****)
PROCEDURE SQUAWK;
BEGIN
  WRITE(CHR(BELL));
END;  (* squawk *)

{------------------------------------------------------------------------------}

(**** blank out lines  ****)
(* given a y location to begin, # of lines to erase, which line *)
(* to leave cursor.  erases only 40 column width.              *)
PROCEDURE BLANKLINES;
VAR I : INTEGER;
BEGIN
  GOTOXY(0,START);
  FOR I := 1 TO (COUNT-1) DO
    WRITELN(' ' : 39);
  WRITE(' ':39);
  GOTOXY(0,ENDCURSOR);
END;   (* blanklines *)

{------------------------------------------------------------------------------}

(* read an acceptable character from the keyboard *)
(* given a set of acceptable characters to read.  *)
(* and flags if you want computer to flush the    *)
(* type ahead buffer, beep if a bad key is pressed*)
(* or echo the character pressed.                 *)
FUNCTION GETCHAR;
VAR MASK : PACKED ARRAY[0..0] OF CHAR;
BEGIN
  IF FLUSHQUEUE THEN UNITCLEAR(2); (* flush buffer *)
  REPEAT
    UNITREAD(2,MASK,1);
    IF BEEP AND NOT (MASK[0] IN OKSET) THEN SQUAWK;
  UNTIL MASK[0] IN OKSET;
  IF ECHO AND (MASK[0] IN [CHR(32)..CHR(126)]) THEN
    WRITE(MASK[0]);
  GETCHAR := MASK[0];
END;  (* getchar *)
```

```
{----------------------------------------------------------------------------}

(**** display a message/wait for a keystroke ****)
PROCEDURE STALL;
VAR STALLCHAR : CHAR;
BEGIN
  WRITE('Press <RET> to continue ');
  STALLCHAR :=
    GETCHAR([CHR(RET),CHR(ESC)],TRUE,FALSE,TRUE);
  IF STALLCHAR = CHR(ESC) THEN EXIT(PROGRAM);
END;  (* stall *)

{----------------------------------------------------------------------------}

(* read in a string and save in a temporary buffer  *)
(* allows control on what characters may be typed   *)
(* and the number of characters typed.              *)
PROCEDURE FILLBUF;
VAR I : INTEGER;
    IDCHAR : CHAR;
BEGIN
  I := 0;
  REPEAT
    IF I > (CHARCNT-1) THEN
      IDCHAR :=
        GETCHAR([CHR(LARROW),CHR(RET)],TRUE,TRUE,TRUE)
    ELSE
    BEGIN {1}
      IDCHAR :=
        GETCHAR(OKSET + [CHR(RET),
                CHR(LARROW),CHR(RARROW)],
                TRUE,TRUE,TRUE);
      IF IDCHAR IN OKSET THEN
      BEGIN {2}
        LINEBUF[I] := IDCHAR;
        I := I + 1;
      END;  {2}
    END;  {1}
    IF IDCHAR = CHR(LARROW) THEN
    BEGIN {3}
      IF I = 0 THEN
        SQUAWK
      ELSE
      BEGIN {4}
        WRITE(CHR(LARROW));
        I := I - 1;
        IF ERASE THEN
        BEGIN {5}
          WRITE(' ');
          WRITE(CHR(LARROW));
          LINEBUF[I] := ' ';
        END;  {5}
      END;  {4}
    END  {3}
    ELSE
      IF IDCHAR = CHR(RARROW) THEN
      BEGIN {6}
        WRITE(LINEBUF[I]);
        I := I + 1;
      END;  {6}
  UNTIL IDCHAR = CHR(RET);
END;  (* fillbuf *)

{----------------------------------------------------------------------------}

(**** open a new text file ****)
PROCEDURE  GETNEWFILE;
VAR FILENAME : STRING;
    ERRNUM : INTEGER;

  {..........................................................................}
  (***** get a legal filename *****)
  FUNCTION NAMEOK : BOOLEAN;
  VAR I : INTEGER;
  BEGIN
    IF FILENAME = ''
      THEN BEGIN {1}
             FILENAME := DEFAULTFILE;
             GOTOXY(44,0);
             WRITE(FILENAME);
           END   {1}
      ELSE IF FILENAME[1] = CHR(esc) THEN EXIT(PROGRAM);
    IF (POS('.TEXT',FILENAME) <> (LENGTH(FILENAME) - 4)) OR
     (LENGTH(FILENAME) < 6 )
       THEN FILENAME := CONCAT(FILENAME,'.TEXT');
  (*$I-*)
    RESET(DEST,FILENAME);
  (*$I+*)
    IF IORESULT = 0
      THEN BEGIN {2}
             WRITELN;
             WRITELN;
             WRITE('Destroy old ',FILENAME,'?  Press ''N'' or ''Y'' ');
             IF GETCHAR(['y','n','Y','N'],TRUE,TRUE,TRUE) IN ['Y','y']
               THEN BEGIN {3}
                      CLOSE(DEST,PURGE);
                      REWRITE(DEST,FILENAME);
                      NAMEOK := TRUE;
                    END   {3}
               ELSE BEGIN {4}
                      CLOSE(DEST,LOCK);
                      NAMEOK := FALSE;
                    END;  {4}
           END   {2}
      ELSE BEGIN {5}
           (*$I-*)
             REWRITE(DEST,FILENAME);
           (*$I+*)
             ERRNUM := IORESULT;
             IF IORESULT <> 0
              THEN BEGIN {6}
                     WRITELN;
                     WRITELN;
                     WRITELN('Cannot open ',FILENAME,' Io error #',ERRNUM);
                     NAMEOK := FALSE;
                   END   {6}
              ELSE NAMEOK := TRUE;
           END;  {5}
  END; (* nameok *)

{..............................................................................}

BEGIN  (* getnewfile  *)
  REPEAT
    PAGE(OUTPUT);
    WRITE('Enter output file name, then press <RET>  : ');
    READLN(FILENAME);
  UNTIL NAMEOK;
END;

{------------------------------------------------------------------------------}

(* send control characters to screen *)
PROCEDURE SCRCONTROL;          { PASCAL interface to Screen Control}
VAR N: INTEGER;                           { APPLE III Standard Device Drivers}
    G_ARRAY:PACKED ARRAY[0.. 3] OF 0..255;          {....... Pages 34 to 46.}
BEGIN
  G_ARRAY[0]:= I; G_ARRAY[1]:=J; G_ARRAY[2]:=K;
  UNITWRITE(1,G_ARRAY,3,,12);
END;   (* scrcontrol *)

{------------------------------------------------------------------------------}

(* switch to 40 column screen *)
```

```
PROCEDURE TEXT40MODE;
BEGIN
  SCRCONTROL(16,0,28);        {Text mode 40BW, followed by clearscreen.}
END;  (* text40mode *)

{-----------------------------------------------------------------------------}

(* switch to 80 column screen *)
PROCEDURE TEXT80MODE;
BEGIN
  SCRCONTROL(04,0,0);       {Restore Viewport to its original condition.}
  SCRCONTROL(16,2,28);      {Text Mode 80, followed by clearscreen.}
END;   (* text80mode *)

{-----------------------------------------------------------------------------}

(* turn on reverse video *)
(* black on white        *)
PROCEDURE INVERSE;
BEGIN
  SCRCONTROL(18,0,0);
END;    (* inverse *)

{-----------------------------------------------------------------------------}

(* switch back to normal screen *)
(* white on black *)
PROCEDURE NORMAL;
BEGIN
  SCRCONTROL(17,0,0);
END;
```

```
(*************************************************************************)
(*                                                                       *)
(*      Textfile : EMGR.DIR/E.SUBRT.TEXT        Volume : TFILES          *)
(*      Codefile : E.MGR.CODE ('Include' file)  Volume : CATDATA         *)
(*                                                                       *)
(*************************************************************************)
(*              DEC. 1, 1982                    NPROC                    *)
(*************************************************************************)

(* initialize  the examinee records *)
PROCEDURE INITEREC;
VAR I : INTEGER;
BEGIN

  (* initialize test taking information *)
  WITH EXAMINEE DO
  BEGIN (1)
    ID := '           ';
    LASTTEST := GMAXSUBTEST; (* mark no tests taken yet *)
    NUMPROC := 0;
    TOTTIMECONSOLE := 0;
    NUMERRORS := 0;
    PREVTIMELASTTEST := 0;
    ORIENTATIONTIME := 0;
  END;  (1)

  (* initialize personal info *)
  WITH PINFO DO
  BEGIN (2)
    LASTNAME := '              ';
    FIRSTNAME := '           ';
    MINITIAL := ' ';
    CURRADDRESS := '   ';
    HOMEOFREC := '  ';
    CITIZENSHIP := '    ';
    SEX := ' ';
    POPGROUP := '    ';
    ETHNIC := '  ';
    MARITAL := ' ';
    DEPENDANTS := '   ';
    BIRTHDATE := '       ';
    EDUCATION := '   ';
    TESTID := '  ';
    AFQT := '  ';
    ASVAB := '                                           ';
    ENLISTDATE := '       ';
    ACTSERDATE := '       ';
    ENL := '    ';
    AFEES := '  ';
    BOS := '  ';
    POSTASVAB := '                                           ';
   STESTORDER := '                                                                ';
  END;  (2)

  (*  initialize the tests results record *)
  FOR I := 0 TO QUESTIONS DO
    TESTS.ITEMINFO[I].ITEMNUM := NIL;
  WITH TESTS DO
  BEGIN (3)
    STTIME := '';
    STINSTRTIME := 0;
    STPROCTCALLS := 0;
    NUMITEMS := 0;
    NUMCORR := 0;
    ESTABILITY := 0;
  END;  (3)
END;   (* initerec *)

(-----------------------------------------------------------------------)

(* get directory index for an id number *)
FUNCTION DIRINDEXNUM;
VAR I : INTEGER;
```

```
    DONE : BOOLEAN;
BEGIN
  LOADINDEX;
  I := 0;
  DONE := FALSE;
  DIRINDEXNUM := NIL;
  REPEAT
    IF (DIR[I].ID = IDNUM)
       AND (NOT (DIR[I].UNUSED)) THEN
    BEGIN {1}
      DONE := TRUE;
      DIRINDEXNUM := I;
    END;  {1}
    I := I + 1;
  UNTIL (I > MAXEXAMINEE) OR (DONE);
END;

{-------------------------------------------------------------------------------}

(* get examinee id number *)
PROCEDURE ENTERID;
VAR INSTRUCTIONS : STRING;
    OKID : BOOLEAN;
BEGIN
  OKID := FALSE;
  REPEAT
    PAGE(OUTPUT);
    FILLCHAR(LINEBUF[0],9,' ');
    GOTOXY(0,10);
    INSTRUCTIONS := 'Enter Examinee Social Security number : ';
    (* read in 9 digit social security number *)
    WRITE(INSTRUCTIONS,'---------  <return>');
    GOTOXY(LENGTH(INSTRUCTIONS),10);
    FILLBUF(9,DIGITS,TRUE);
    MOVELEFT(LINEBUF[0],EXAMINEE.ID[0],9);
    IF EXAMINEE.ID = '         ' THEN
      SQUAWK
    ELSE
      OKID := TRUE;
  UNTIL OKID;
  FILLCHAR(LINEBUF[0],9,' ');
END;  (* enter id *)
```

```
(****************************************************************************)
(*                                                                          *)
(*      Textfile : EMGR.DIR/E.IO.TEXT              Volume : TFILES          *)
(*      Codefile : E.MGR.CODE ('Include' file)     Volume : CATDATA         *)
(*                                                                          *)
(****************************************************************************)
(*              DEC.  1, 1982                      NPROC                    *)
(****************************************************************************)

(* loads examinee personal data *)
PROCEDURE LOADPDATA;
BEGIN
  RESET(PINFOFILE,PINFONAME);
  SEEK(PINFOFILE,RECORDNUM);
  GET(PINFOFILE);
  PINFO := PINFOFILE^;
  CLOSE(PINFOFILE,LOCK);
END;   (* load personal data *)

{--------------------------------------------------------------------------}

(* saves examinee personal data *)
PROCEDURE SAVEPDATA;
BEGIN
  RESET(PINFOFILE,PINFONAME);
  SEEK(PINFOFILE,RECORDNUM);
  PINFOFILE^ := PINFO;
  PUT(PINFOFILE);
  CLOSE(PINFOFILE,LOCK);
END;   (* load personal data *)


{--------------------------------------------------------------------------}

(* loads examinee directory *)
PROCEDURE LOADINDEX;
BEGIN
  RESET(FILEDIR,INDEXNAME);
  SEEK(FILEDIR,0);
  GET(FILEDIR);
  DIR := FILEDIR^;
  CLOSE(FILEDIR,LOCK);
END;

{--------------------------------------------------------------------------}

(* loads examinee personal data *)
PROCEDURE LOADEXAMINEE;
BEGIN
  RESET(FILEEXAMINEE,INFONAME);
  SEEK(FILEEXAMINEE,RECNUM);
  GET(FILEEXAMINEE);
  EXAMINEE := FILEEXAMINEE^;
  CLOSE(FILEEXAMINEE,LOCK);
END;

{--------------------------------------------------------------------------}

(* updates examinee personal data record *)
PROCEDURE UPDATEEXAMINEE;
BEGIN
  RESET(FILEEXAMINEE,INFONAME);
  SEEK(FILEEXAMINEE,RECNUM);
  FILEEXAMINEE^ := EXAMINEE;
  PUT(FILEEXAMINEE);
  CLOSE(FILEEXAMINEE,LOCK);
END;

{--------------------------------------------------------------------------}

(* updates examinee directory *)
PROCEDURE UPDATEINDEX;
BEGIN
```

```
  RESET(FILEDIR,INDEXNAME);
  SEEK(FILEDIR,0);
  FILEDIR^ := DIR;
  PUT(FILEDIR);
  CLOSE(FILEDIR,LOCK);
END;

{-----------------------------------------------------------------------------}

(* loads examines test scores *)
PROCEDURE LOADRESULTS;
BEGIN
  RESET(FILETESTS,RESULTS);
  SEEK(FILETESTS,RECNUM);
  GET(FILETESTS);
  TESTS := FILETESTS^;
  CLOSE(FILETESTS,LOCK);
END;

{-----------------------------------------------------------------------------}

(* updates examinee test scores *)
PROCEDURE UPDATERESULTS;
BEGIN
  RESET(FILETESTS,RESULTS);
  SEEK(FILETESTS,RECNUM);
  FILETESTS^ := TESTS;
  PUT(FILETESTS);
  CLOSE(FILETESTS,LOCK);
END;
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : EMGR.DIR/E.LOGIN.TEXT          Volume : TFILES             *)
(*      Codefile : E.MGR.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(*               DEC. 14, 1982                 NPROC                          *)
(******************************************************************************)

(* log in the examinee, put them into the system *)
PROCEDURE LOGINEXAMINEE;

VAR ERECNUM : INTEGER;
    EID : IDTYPE;
    DONE : BOOLEAN;
    CHOICE : CHAR;


    (* locate unused slot in directory *)
    (* returns nil if no space left.   *)
    FUNCTION OPENSLOT : INTEGER;
    VAR I : INTEGER;
        DONE : BOOLEAN;
    BEGIN
      I := 0;
      REPEAT
        IF DIR[I].UNUSED
          THEN DONE := TRUE
          ELSE I := I + 1;
      UNTIL (I > MAXEXAMINEE) OR (DONE);
      IF DONE
        THEN OPENSLOT := I
        ELSE OPENSLOT := NIL;
    END;

    {-----------------------------------------------------------------------}

    (* get directory index for an id number *)
    (* returns nil if no such id exists.    *)
    FUNCTION DIRINDEXNUM(IDNUM : IDTYPE) : INTEGER;
    VAR I : INTEGER;
        DONE : BOOLEAN;
    BEGIN
      LOADINDEX;
      I := 0;
      DONE := FALSE;
      DIRINDEXNUM := NIL;
      REPEAT
        IF (DIR[I].ID = IDNUM) AND
           (NOT (DIR[I].UNUSED)) THEN
        BEGIN {1}
          DONE := TRUE;
          DIRINDEXNUM := I;
        END;  {1}
        I := I + 1;
      UNTIL (I > MAXEXAMINEE) OR (DONE);
    END;  (* dirindexnum *)

{.........................................................................}

BEGIN   (* log in *)
  (* initialize line buffer *)
  FILLCHAR(LINEBUF[0],79,' ');

  DONE := FALSE;
  REPEAT
    ENTERID;  (* get id number *)
    EID := EXAMINEE.ID;
    ERECNUM := DIRINDEXNUM(EXAMINEE.ID);
    IF ERECNUM >= 0 THEN    (* examinee exists in directory already *)
    BEGIN {1}
      LOADEXAMINEE(ERECNUM);
      WRITELN;
      WRITELN;
      WRITELN('Examinee ',EXAMINEE.ID,' previously logged in.');
      WRITELN;
      IF EXAMINEE.LASTTEST > GMAXSUBTEST THEN
        WRITELN('This examinee has completed all testing.');
      WRITELN;
      STALL;
    END   {1}
    ELSE
    BEGIN {2}
      (* find open directory/set record # to store examinee *)
      ERECNUM := OPENSLOT;
      IF ERECNUM < 0 THEN
      BEGIN {3}
        PAGE(OUTPUT);
        GOTOXY(0,10);
        WRITELN('No room in directory put examinee!');
        WRITELN;
        WRITELN;
        STALL;
        EXIT(LOGINEXAMINEE);
      END   {3}
      ELSE
      BEGIN {4}
        LOADEXAMINEE(ERECNUM);
        DIR[ERECNUM].UNUSED := FALSE;
        DIR[ERECNUM].ID := EID;
        EXAMINEE.NUMPROC := 0;
        EXAMINEE.ID := EID;
        EXAMINEE.LASTTEST := GMAXSUBTEST;  (* flag new examinee *)

        (* update examinee record on disk *)
        UPDATEEXAMINEE(ERECNUM);

        (* update examinee directory *)
        UPDATEINDEX;

        WRITELN;
        WRITELN;
        WRITELN('Examinee ',EXAMINEE.ID,' entered into system.');
        WRITELN;
        WRITELN;
        WRITELN('Press <RET> to continue logging in examinees or');
        WRITE('press <ESC> to quit. ');
        CHOICE := GETCHAR([CHR(RET),CHR(ESC)],TRUE,FALSE,TRUE);
        IF CHOICE = CHR(ESC) THEN
          DONE := TRUE;
      END;  {4}
    END;  {2}
  UNTIL DONE;

END; (* log-in *)
```

```
(****************************************************************************)
(*                                                                          *)
(*      Textfile : EMGR.DIR/E.FETCH.TEXT          Volume : TFILES           *)
(*      Codefile : E.MGR.CODE ('Include' file)  Volume : CATDATA           *)
(*                                                                          *)
(****************************************************************************)
(* File Last Modified :  Oct 1, 1983                         NPROC         *)
(****************************************************************************)

(* enter / modify  examinee data *)
PROCEDURE ENTERDATA;
VAR IDINDEX : INTEGER;
    COMMAND : CHAR;
    OK1,OK2,OK3,OK4,
    CHANGED,
    FOREVER : BOOLEAN;

    {-----------------------------------------------------------------------}

    (* display examinee personal data *)
    PROCEDURE DISPLAYEXAMINEE;
    VAR J : INTEGER;

        {...................................................................}

        (* display some data *)
        PROCEDURE D1;
        BEGIN
          WITH PINFO DO
          BEGIN {1}
            GOTOXY(0,4);
            WRITE(' Last Name');
            WRITE(' : ',LASTNAME);
            GOTOXY(32,4);
            WRITE('First Name');
            WRITE(' : ',FIRSTNAME);
            GOTOXY(60,4);
            WRITE('Middle Initial');
            WRITE(' : ',MINITIAL);
            GOTOXY(1,5);
            WRITE('Current Address (State)');
            WRITE(' : ',CURRADDRESS);
            GOTOXY(40,5);
            WRITE('Home of Record (State)');
            WRITELN(' : ',HOMEOFREC);
            WRITE(' Citizenship');
            WRITELN(' : ',CITIZENSHIP);
            WRITE(' Sex');
            WRITELN(' : ',SEX);
            WRITE(' Population Group');
            WRITELN(' : ',POPGROUP);
            WRITE(' Ethnic Group');
            WRITELN(' : ',ETHNIC);
            WRITE(' Marital Status');
            WRITELN(' : ',MARITAL);
            WRITE(' Number of Dependents');
            WRITELN(' : ',DEPENDANTS);
            WRITE(' Date of Birth');
            WRITELN(' : ',BIRTHDATE);
          END;  {1}
        END;  (* d1 *)

    {.......................................................................}

    BEGIN  (* display examinee *)
      PAGE(OUTPUT);
      D1;
      WITH PINFO DO
      BEGIN {1}
        WRITE(' Education');
        WRITELN(' : ',EDUCATION);
        WRITE(' Test Id');
        WRITELN(' : ',TESTID);
      WRITE(' AFQT');
      WRITELN(' : ',AFQT);
      WRITE(' ASVAB Scores');
      WRITELN(' : ',ASVAB);
      WRITE(' Date of Enlistment');
      WRITELN(' : ',ENLISTDATE);
      WRITE(' Active Service Date'),
      WRITELN(' : ',ACTSERDATE);
      WRITE(' Rating / MOS');
      WRITELN(' : ',ENL);
      WRITE(' AFEES');
      WRITELN(' : ',AFEES);
      WRITE(' Branch of Service');
      WRITELN(' : ',BOS);
      WRITE(' Retest ASVAB');
      WRITELN(' : ',POSTASVAB);
      WRITE(' Test Order');
      WRITE(' : ',STESTORDER[0],STESTORDER[1]);
    END;  {1}
  END;  (* display examinee *)



  (*-----------------------------------------------------*)


  (* get examinee personal data *)
  PROCEDURE GETDATA;

     {...........................................................}

     (* read in a string and save in a temporary buffer  *)
     PROCEDURE EFILLBUF(CHARCNT : INTEGER;
                        OKSET : SETOFCHAR; ERASE : BOOLEAN);
     VAR I : INTEGER;
         IDCHAR : CHAR;
     BEGIN
       CHANGED := FALSE;
       I := 0;
       REPEAT
         IF I > (CHARCNT-1) THEN
           IDCHAR :=
             GETCHAR([CHR(LARROW),CHR(RET)],TRUE,TRUE,TRUE)
         ELSE
         BEGIN {1}
           IF I = 0 THEN
             IDCHAR :=
               GETCHAR(OKSET + [CHR(RET),
                       CHR(ESC),CHR(LARROW),CHR(RARROW)],
                       TRUE,TRUE,TRUE)
           ELSE
             IDCHAR :=
               GETCHAR(OKSET + [CHR(RET),
                       CHR(LARROW),CHR(RARROW)],
                       TRUE,TRUE,TRUE);

           IF IDCHAR = CHR(ESC) THEN
           BEGIN {2}
             FOREVER := FALSE;
             EXIT(GETDATA);
           END;  {2}

           IF IDCHAR IN OKSET THEN
           BEGIN {3}
             CHANGED := TRUE;
             LINEBUF[I] := IDCHAR;
             I := I + 1;
           END;  {3}
         END;  {1}
         IF IDCHAR = CHR(LARROW) THEN
         BEGIN {4}
           IF I = 0 THEN
    SQUAWK
   ELSE
   BEGIN {5}
    WRITE(CHR(LARROW));
    I := I - 1;
    IF ERASE THEN
    BEGIN {6}
     WRITE(' ');
     WRITE(CHR(LARROW));
     LINEBUF[I] := ' ';
    END;  {6}
   END;  {5}
  END   {4}
  ELSE
   IF IDCHAR = CHR(RARROW) THEN
   BEGIN {7}
    WRITE(LINEBUF[I]);
    I := I + 1;
   END;  {7}
 UNTIL IDCHAR = CHR(RET);
END;  (* Efillbuf *)

{---------------------------------------------------------------------}

(* get personal data part 1 *)
PROCEDURE GET1DATA;
BEGIN
 INVERSE;
 GOTOXY(1,4);
 WRITE('Last Name');
 NORMAL;
 GOTOXY(13,4);
 MOVELEFT(PINFO.LASTNAME[0],LINEBUF[0],15);
 EFILLBUF(15,LETTERS + [' '],TRUE);
 IF CHANGED THEN
  OK1 := FALSE;
 MOVELEFT(LINEBUF[0],PINFO.LASTNAME[0],15);
 FILLCHAR(LINEBUF[0],15,' ');
 GOTOXY(1[illegible]
 WRITE('Last Name');

 INVERSE;
 GOTOXY(32,4);
 WRITE('First Name');
 NORMAL;
 GOTOXY(45,4);
 MOVELEFT(PINFO.FIRSTNAME[0],LINEBUF[0],12);
 EFILLBUF(12,LETTERS + [' '],TRUE);
 MOVELEFT(LINEBUF[0],PINFO.FIRSTNAME[0],12);
 FILLCHAR(LINEBUF[0],12,' ');
 GOTOXY(32,4);
 WRITE('First Name');

 INVERSE;
 GOTOXY(60,4);
 WRITE('Middle Initial');
 NORMAL;
 GOTOXY(77,4);
 LINEBUF[0] := PINFO.MINITIAL;
 EFILLBUF(1,LETTERS + [' '],TRUE);
 PINFO.MINITIAL := LINEBUF[0];
 LINEBUF[0] := ' ';
 GOTOXY(60,4);
 WRITE('Middle Initial');

 INVERSE;
 GOTOXY(1,5);
 WRITE('Current Address (State)');
 NORMAL;
 GOTOXY(27,5);
 MOVELEFT(PINFO.CURRADDRESS[0],LINEBUF[0],2);
 EFILLBUF(2,LETTERS + [' '],TRUE);
 MOVELEFT(LINEBUF[0],PINFO.CURRADDRESS[0],2);
```

```
    FILLCHAR(LINEBUF[0],2,' ');
    GOTOXY(1,5);
    WRITE('Current Address (State)');

    INVERSE;
    GOTOXY(40,5);
    WRITE('Home of Record (State)');
    NORMAL;
    GOTOXY(65,5);
    MOVELEFT(PINFO.HOMEOFREC[0],LINEBUF[0],2);
    EFILLBUF(2,LETTERS + [' '],TRUE);
    MOVELEFT(LINEBUF[0],PINFO.HOMEOFREC[0],2);
    FILLCHAR(LINEBUF[0],2,' ');
    GOTOXY(40,5);
    WRITE('Home of Record (State)');
  END;    (* get1data *)

  {-----------------------------------------------------------------}

  (*  get more personal data *)
  PROCEDURE GET2DATA;
  BEGIN
    INVERSE;
    GOTOXY(1,6);
    WRITE('Citizenship');
    NORMAL;
    GOTOXY(15,6);
    MOVELEFT(PINFO.CITIZENSHIP[0],LINEBUF[0],4);
    EFILLBUF(4,LETTERS + [' '],TRUE);
    MOVELEFT(LINEBUF[0],PINFO.CITIZENSHIP[0],4);
    FILLCHAR(LINEBUF[0],4,' ');
    GOTOXY(1,6);
    WRITE('Citizenship');

    INVERSE;
    GOTOXY(1,7);
    WRITE('Sex');
    NORMAL;
    GOTOXY(7,7);
    LINEBUF[0] := PINFO.SEX;
    EFILLBUF(1,LETTERS + [' '],TRUE);
    PINFO.SEX := LINEBUF[0];
    LINEBUF[0] := ' ';
    GOTOXY(1,7);
    WRITE('Sex');

    INVERSE;
    GOTOXY(1,8);
    WRITE('Population Group');
    NORMAL;
    GOTOXY(20,8);
    MOVELEFT(PINFO.POPGROUP[0],LINEBUF[0],1);
    EFILLBUF(1,DIGITS + [' '] + LETTERS,TRUE);
    MOVELEFT(LINEBUF[0],PINFO.POPGROUP[0],1);
    FILLCHAR(LINEBUF[0],1,' ');
    GOTOXY(1,8);
    WRITE('Population Group');

    INVERSE;
    GOTOXY(1,9);
    WRITE('Ethnic Group');
    NORMAL;
    GOTOXY(16,9);
    MOVELEFT(PINFO.ETHNIC[0],LINEBUF[0],1);
    EFILLBUF(1,LETTERS + DIGITS + [' '],TRUE);
    MOVELEFT(LINEBUF[0],PINFO.ETHNIC[0],1);
    FILLCHAR(LINEBUF[0],1,' ');
    GOTOXY(1,9);
    WRITE('Ethnic Group');

    INVERSE;
    GOTOXY(1,10);
  WRITE('Marital Status');
  NORMAL;
  GOTOXY(18,10);
  LINEBUF[0] := PINFO.MARITAL;
  EFILLBUF(1,LETTERS + [' '],TRUE);
  PINFO.MARITAL := LINEBUF[0];
  LINEBUF[0] := ' ';
  GOTOXY(1,10);
  WRITE('Marital Status');
END;   (* get2data *)

 {---------------------------------------------------------------------------}

 (* get personal data part 3 *)
PROCEDURE GET3DATA;
BEGIN
  INVERSE;
  GOTOXY(1,11);
  WRITE('Number of Dependents');
  NORMAL;
  GOTOXY(24,11);
  MOVELEFT(PINFO.DEPENDANTS[0],LINEBUF[0],2);
  EFILLBUF(2,DIGITS + [' '],TRUE);
  MOVELEFT(LINEBUF[0],PINFO.DEPENDANTS[0],2);
  FILLCHAR(LINEBUF[0],2,' ');
  GOTOXY(1,11);
  WRITE('Number of Dependents');

  INVERSE;
  GOTOXY(1,12);
  WRITE('Date of Birth');
  NORMAL;
  GOTOXY(17,12);
  MOVELEFT(PINFO.BIRTHDATE[0],LINEBUF[0],8);
  EFILLBUF(8,DIGITS + ['/',' '],TRUE);
  MOVELEFT(LINEBUF[0],PINFO.BIRTHDATE[0],8);
  FILLCHAR(LINEBUF[0],8,' ');
  GOTOXY(1,12);
  WRITE('Date of Birth');

  INVERSE;
  GOTOXY(1,13);
  WRITE('Education');
  NORMAL;
  GOTOXY(13,13);
  MOVELEFT(PINFO.EDUCATION[0],LINEBUF[0],3);
  EFILLBUF(3,LETTERS + DIGITS + [' '],TRUE);
  MOVELEFT(LINEBUF[0],PINFO.EDUCATION[0],3);
  FILLCHAR(LINEBUF[0],3,' ');
  GOTOXY(1,13);
  WRITE('Education');

  INVERSE;
  GOTOXY(1,14);
  WRITE('Test Id');
  NORMAL;
  GOTOXY(11,14);
  MOVELEFT(PINFO.TESTID[0],LINEBUF[0],3);
  EFILLBUF(3,DIGITS + LETTERS + [' '],TRUE);
  MOVELEFT(LINEBUF[0],PINFO.TESTID[0],3);
  FILLCHAR(LINEBUF[0],3,' ');
  GOTOXY(1,14);
  WRITE('Test Id');

  INVERSE;
  GOTOXY(1,15);
  WRITE('AFQT');
  NORMAL;
  GOTOXY(8,15);
  MOVELEFT(PINFO.AFQT[0],LINEBUF[0],2);
  EFILLBUF(4,DIGITS + [' '],TRUE);
  MOVELEFT(LINEBUF[0],PINFO.AFQT[0],2);
  FILLCHAR(LINEBUF[0],2,' ');
  GOTOXY(1,15);
  WRITE('AFQT');
END;   (* get3data *)

{-------------------------------------------------------------------}

(*  get more personal data part 4 *)
PROCEDURE GET4DATA;
BEGIN
  INVERSE;
  GOTOXY(1,16);
  WRITE('ASVAB Scores');
  NORMAL;
  GOTOXY(16,16);
  MOVELEFT(PINFO.ASVAB[0],LINEBUF[0],32);
  EFILLBUF(32,DIGITS + [' '],TRUE);
  IF CHANGED THEN
    OK2 := FALSE;
  MOVELEFT(LINEBUF[0],PINFO.ASVAB[0],32);
  FILLCHAR(LINEBUF[0],32,' ');
  GOTOXY(1,16);
  WRITE('ASVAB Scores');

  INVERSE;
  GOTOXY(1,17);
  WRITE('Date of Enlistment');
  NORMAL;
  GOTOXY(22,17);
  MOVELEFT(PINFO.ENLISTDATE[0],LINEBUF[0],8);
  EFILLBUF(8,DIGITS + ['/',' '],TRUE);
  MOVELEFT(LINEBUF[0],PINFO.ENLISTDATE[0],8);
  FILLCHAR(LINEBUF[0],8,' ');
  GOTOXY(1,17);
  WRITE('Date of Enlistment');

  INVERSE;
  GOTOXY(1,18);
  WRITE('Active Service Date');
  NORMAL;
  GOTOXY(23,18);
  MOVELEFT(PINFO.ACTSERDATE[0],LINEBUF[0],8);
  EFILLBUF(8,DIGITS + ['/',' '],TRUE);
  MOVELEFT(LINEBUF[0],PINFO.ACTSERDATE[0],8);
  FILLCHAR(LINEBUF[0],8,' ');
  GOTOXY(1,18);
  WRITE('Active Service Date');

  INVERSE;
  GOTOXY(1,19);
  WRITE('Rating / MOS');
  NORMAL;
  GOTOXY(16,19);
  MOVELEFT(PINFO.ENL[0],LINEBUF[0],5);
  EFILLBUF(5,DIGITS + LETTERS + [' '],TRUE);
  IF CHANGED THEN
    OK3 := FALSE;
  MOVELEFT(LINEBUF[0],PINFO.ENL[0],5);
  FILLCHAR(LINEBUF[0],5,' ');
  GOTOXY(1,19);
  WRITE('Rating / MOS');

  INVERSE;
  GOTOXY(1,20);
  WRITE('AFEES');
  NORMAL;
  GOTOXY(9,20);
  MOVELEFT(PINFO.AFEES[0],LINEBUF[0],2);
  EFILLBUF(2,LETTERS + DIGITS + [' '],TRUE);
  MOVELEFT(LINEBUF[0],PINFO.AFEES[0],2);
  FILLCHAR(LINEBUF[0],2,' ');
  GOTOXY(1,20);
  WRITE('AFEES');
```

```
       END;    (* get4data  *)

       (* --------------------------------------------------------------- *)


       (* get some more data *)
       PROCEDURE GET5DATA;
       BEGIN
         INVERSE;
         GOTOXY(1,21);
         WRITE('Branch of Service');
         NORMAL;
         GOTOXY(21,21);
         MOVELEFT(PINFO.BOS[0],LINEBUF[0],2);
         EFILLBUF(2,LETTERS + [' '],TRUE);
         MOVELEFT(LINEBUF[0],PINFO.BOS[0],2);
         FILLCHAR(LINEBUF[0],2,' ');
         GOTOXY(1,21);
         WRITE('Branch of Service');

         INVERSE;
         GOTOXY(1,22);
         WRITE('Retest ASVAB');
         NORMAL;
         GOTOXY(16,22);
         MOVELEFT(PINFO.POSTASVAB[0],LINEBUF[0],32);
         EFILLBUF(32,DIGITS + [' '],TRUE);
         IF CHANGED THEN
           OK4 := FALSE;
         MOVELEFT(LINEBUF[0],PINFO.POSTASVAB[0],32);
         FILLCHAR(LINEBUF[0],32,' ');
         GOTOXY(1,22);
         WRITE('Retest ASVAB');

         INVERSE;
         GOTOXY(1,23);
         WRITE('Test Order');
         NORMAL;
         GOTOXY(14,23);
         MOVELEFT(PINFO.stestorder[0],LINEBUF[0],2);
         EFILLBUF(2,LETTERS + [' '],TRUE);
         MOVELEFT(LINEBUF[0],PINFO.STESTORDER[0],2);
         FILLCHAR(LINEBUF[0],2,' ');
         GOTOXY(1,23);
         WRITE('Test Order');

       END;  (* get5data *)

    (* ------------------------ *)

    BEGIN  (* getdata *)
      GOTOXY(0,0);
      WRITELN(
'DATA ENTRY INSTRUCTIONS : To enter/modify data, type in the data you want');
      WRITELN(
'and then press <RET> to accept that data. To skip the current field, just');
      WRITELN(
'press <RET>.  To quit, press <RET> and then <ESC>.');

      FOREVER := TRUE;
      REPEAT
        GET1DATA;
        GET2DATA;
        GET3DATA;
        GET4DATA;
        GET5DATA;
      UNTIL NOT FOREVER;
    END;  (* getdata *)
    (* -------------------------------------------------------------- *)
(*  verify certain data *)
PROCEDURE VERIFYDATA;
VAR VLASTNAME : PACKED ARRAY[0..14] OF CHAR;
    VASVAB,VRETEST : PACKED ARRAY[0..43] OF CHAR;
    VENL : PACKED ARRAY[0..4] OF CHAR;

    DONE : BOOLEAN;


    (* get verified data *)
    PROCEDURE GETVDATA;
    BEGIN
      FILLCHAR(LINEBUF[0],80,' ');

      GOTOXY(1,4);
      WRITE('Last Name : ');
      IF OK1 THEN WRITE(PINFO.LASTNAME,'    (Data OK)');
      GOTOXY(1,5);
      WRITE('ASVAB Scores : ');
      IF OK2 THEN WRITE(PINFO.ASVAB,'    (Data OK)');
      GOTOXY(1,6);
      WRITE('Rating / MOS : ');
      IF OK3 THEN WRITE(PINFO.ENL,'    (Data OK)');
      GOTOXY(1,7);
      WRITE('Retest ASVAB Scores : ');
      IF OK4 THEN WRITE(PINFO.POSTASVAB,'    (Data OK)');

      IF NOT OK1 THEN
      BEGIN
        INVERSE;
        GOTOXY(1,4);
        WRITE('Last Name');
        NORMAL;
        WRITE(' : ');
        GOTOXY(13,4);
        FILLBUF(15,LETTERS + [' '],TRUE);
        MOVELEFT(LINEBUF[0],VLASTNAME[0],15);
        FILLCHAR(LINEBUF[0],15,' ');
        GOTOXY(1,4);
        WRITE('Last Name');
      END;

      IF NOT OK2 THEN
      BEGIN
        INVERSE;
        GOTOXY(1,5);
        WRITE('ASVAB Scores');
        NORMAL;
        WRITE(' : ');
        GOTOXY(16,5);
        FILLBUF(32,DIGITS + [' '],TRUE);
        MOVELEFT(LINEBUF[0],VASVAB[0],32);
        FILLCHAR(LINEBUF[0],32,' ');
        GOTOXY(1,5);
        WRITE('ASVAB Scores');
      END;

      IF NOT OK3 THEN
      BEGIN
        INVERSE;
        GOTOXY(1,6);
        WRITE('Rating / MOS');
        NORMAL;
        WRITE(' : ');
        GOTOXY(16,6);
        FILLBUF(5,DIGITS + LETTERS + [' '],TRUE);
        MOVELEFT(LINEBUF[0],VENL[0],5);
        FILLCHAR(LINEBUF[0],5,' ');
        GOTOXY(1,6);
        WRITE('Rating / MOS');
      END;
      IF NOT OK4 THEN
      BEGIN
        INVERSE;
        GOTOXY(1,7);
        WRITE('Retest ASVAB Scores');
        NORMAL;
        WRITE(' : ');
        GOTOXY(23,7);
        FILLBUF(32,DIGITS + [' '],TRUE);
        MOVELEFT(LINEBUF[0],VRETEST[0],32);
        FILLCHAR(LINEBUF[0],32,' ');
        GOTOXY(1,7);
        WRITE('Retest ASVAB Scores');
      END;

    END;  (* getvdata *)


    (*---------------------------------------------------------*)


    PROCEDURE CHECKDAT;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(0,4);
      IF (PINFO.LASTNAME <> VLASTNAME) THEN
        WRITELN('*** ERROR ***')
      ELSE
        OK1 := TRUE;
      WRITELN('Previous Last Name : ',PINFO.LASTNAME);
      WRITELN('Verified Last Name : ',VLASTNAME);
      WRITELN;
      IF (PINFO.ASVAB <> VASVAB) THEN
        WRITELN('*** ERROR ***')
      ELSE
        OK2 := TRUE;
      WRITELN('Previous ASVAB Scores : ',PINFO.ASVAB);
      WRITELN('Verified ASVAB Scores : ',VASVAB);
      WRITELN;
      IF (PINFO.ENL <> VENL) THEN
        WRITELN('*** ERROR ***')
      ELSE
        OK3 := TRUE;
      WRITELN('Previous Rating / MOS : ',PINFO.ENL);
      WRITELN('Verified Rating / MOS : ',VENL);
      WRITELN;
      IF (PINFO.POSTASVAB <> VRETEST) THEN
        WRITELN('*** ERROR ***')
      ELSE
        OK4 := TRUE;
      WRITELN('Previous Retest ASVAB Scores : ',PINFO.POSTASVAB);
      WRITELN('Verified Retest ASVAB Scores : ',VRETEST);
      WRITELN;
      WRITELN;
    END;  (* checkdat *)


BEGIN    (* verifydata *)
  IF  OK1 THEN
    VLASTNAME := PINFO.LASTNAME
  ELSE
    FILLCHAR(VLASTNAME[0],15,' ');
  IF  OK2 THEN
    VASVAB := PINFO.ASVAB
  ELSE
    FILLCHAR(VASVAB[0],44,' ');
  IF OK3 THEN
    VENL := PINFO.ENL
  ELSE
    FILLCHAR(VENL[0],5,' ');
  IF OK4 THEN
    VRETEST := PINFO.POSTASVAB
```

```
      ELSE
        FILLCHAR(VRETEST[0],44,' ');
      DONE := FALSE;
      REPEAT
        PAGE(OUTPUT);
        WRITELN(
'To verify certain pieces of data, please enter the information below for the');
        WRITELN(
'examinee again.');

        GETVDATA;
        CHECKDAT;

        IF (PINFO.ENL <> VENL) OR
          (PINFO.ASVAB <> VASVAB) OR
          (PINFO.LASTNAME <> VLASTNAME) OR
          (PINFO.POSTASVAB <> VRETEST) THEN
        BEGIN
          SQUAWK;
          WRITE('Do you want to re-enter the data?   Y/N  : ');
          IF GETCHAR(['Y','N','y','n'],TRUE,TRUE,TRUE) IN ['Y','y'] THEN
          BEGIN
            DISPLAYEXAMINEE;
            GETDATA;
          END
          ELSE
          BEGIN
            DONE := TRUE;
            IF NOT OK1 THEN FILLCHAR(PINFO.LASTNAME[0],15,' ');
            IF NOT OK2 THEN FILLCHAR(PINFO.ASVAB[0],44,' ');
            IF NOT OK3 THEN FILLCHAR(PINFO.ENL[0],5,' ');
            IF NOT OK4 THEN FILLCHAR(PINFO.POSTASVAB[0],44,' ');
          END;
        END
        ELSE
        BEGIN
          DONE := TRUE;
          WRITE('Verified data is OK.');
          WRITELN;
          WRITELN;
          STALL;
        END;
      UNTIL DONE;
    END;  (* verifydata *)

{.............................................................................}

BEGIN (* enter data *)
  LOADINDEX;
  ENTERID;
  IDINDEX := DIRINDEXNUM(EXAMINEE.ID);
  IF IDINDEX < 0 THEN
  BEGIN {1}
    GOTOXY(0,12);
    WRITELN('No record in file with ID : ',EXAMINEE.ID);
    WRITELN;
    STALL;
  END   {1}
  ELSE
  BEGIN {2}
    LOADPDATA(IDINDEX);
    DISPLAYEXAMINEE;
    OK1 := TRUE;
    OK2 := TRUE;
    OK3 := TRUE;
    OK4 := TRUE;
    GETDATA;
    VERIFYDATA;
    SAVEPDATA(IDINDEX);
  END;  {2}
END;  (* enter data *)
```

```
(*****************************************************************************)
(*                                                                           *)
(*      Textfile : EMGR.DIR/E.STATUS.TEXT       Volume : TFILES              *)
(*      Codefile : E.MGR.CODE ('Include' file)  Volume : CATDATA             *)
(*                                                                           *)
(*****************************************************************************)
(*              DEC.  1, 1982                   NPRDC                        *)
(*****************************************************************************)

(* list the status of all examinees in directory *)

PROCEDURE ESTATUS;
VAR COUNT,
    STATUSCODE,
    I,K : INTEGER;
    IDCHAR : CHAR;

BEGIN
  PAGE(OUTPUT);
  WRITELN;
  WRITELN(' Examinee Id #                Status');
  WRITELN('------------------------------------------------------------------');
  WRITELN;
  LOADINDEX;
  COUNT := 0;
  FOR K := 0 TO 9 DO
  BEGIN (1)
    IDCHAR := CHR(K+48);
    FOR I := 0 TO MAXEXAMINEE DO
    BEGIN (2)
      IF NOT (DIR[I].UNUSED)  THEN
      BEGIN (3)
        IF DIR[I].ID[0] = IDCHAR THEN
        BEGIN (4)
          COUNT := COUNT + 1;
          WRITE(DIR[I].ID);
          WRITE('                        ');
          LOADEXAMINEE(I);
          STATUSCODE := EXAMINEE.LASTTEST;
          IF STATUSCODE < GMAXSUBTEST THEN
            WRITELN('Session not completed, ',EXAMINEE.LASTTEST,
                    ' tests finished.');
          IF STATUSCODE = GMAXSUBTEST THEN
            WRITELN('New examinee');
          IF STATUSCODE > GMAXSUBTEST THEN
            WRITELN('Examinee has completed session.');

        END;  (4)
      END;  (3)
    END;  (2)
    WRITELN;
  END;  (1)
  WRITELN;
  WRITELN('------------------------------------------------------------------');
  WRITELN;
  WRITELN(COUNT,' examinees in directory.');
  WRITELN;
  STALL;
END;   (* estatus *)
```

```
(****************************************************************************)
(*                                                                          *)
(*      Textfile : EMGR.DIR/E.DELETE.TEXT       Volume : TFILES             *)
(*      Codefile : E.MGR.CODE ('Include' file)  Volume : CATDATA            *)
(*                                                                          *)
(****************************************************************************)
(*                DEC.  1, 1982                 NPROC                       *)
(****************************************************************************)

(* remove examinee record from file *)
PROCEDURE REMOVEEXAMINEE(IDNUM : INTEGER);
VAR TESTSLOT,
    J  : INTEGER;
BEGIN

  (* initialize the examinee records *)
  INITEREC;

  (* zero out test taking info *)
  UPDATEEXAMINEE(IDNUM);

  (* zero out personal data *)
  SAVEPDATA(IDNUM);

  (* zero out testing results *)
  TESTSLOT := IDNUM * STESTS;
  RESET(FILETESTS,RESULTS);
  J := 0;
  REPEAT
    SEEK(FILETESTS,TESTSLOT);
    FILETESTS^ := TESTS;
    PUT(FILETESTS);
    TESTSLOT := TESTSLOT + 1;
    J := J + 1;
  UNTIL J = STESTS;
  CLOSE(FILETESTS,LOCK);

  (* update directory with examinee removed *)
  DIR[IDNUM].UNUSED := TRUE;
  UPDATEINDEX;
END;  (* remove examinee *)

{---------------------------------------------------------------------------}

(**** get record in a file to delete ****)
PROCEDURE DELETEEXAMINEE;
VAR IDINDEX : INTEGER;
BEGIN  (* delete  *)
  LOADINDEX;
  ENTERID;
  IDINDEX := DIRINDEXNUM(EXAMINEE.ID);
  IF IDINDEX < 0 THEN
  BEGIN (1)
    GOTOXY(1,12);
    WRITELN('No record in file with ID : ',EXAMINEE.ID);
    WRITELN;
    STALL;
  END   (1)
  ELSE
  BEGIN (2)
    WRITELN;
    WRITELN;
    WRITE('Delete  examinee ',EXAMINEE.ID,' ?  (Press ''N'' or ''Y'') : ');
    IF GETCHAR(['y','n','Y','N'],TRUE,TRUE,TRUE) IN ['Y','y'] THEN
      REMOVEEXAMINEE(IDINDEX);
  END;  (2)
END;  (* delete examinee *)
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : EMGR.DIR/E.ENDOFDAY.TEXT     Volume : TFILES               *)
(*      Codefile : E.MGR.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(*              DEC.  1, 1982                   NPROC                         *)
(******************************************************************************)

(* back up all examinees who have finished session *)

PROCEDURE ENDOFDAY;

VAR COUNT,
    STATUSCODE,
    I,J,K : INTEGER;

    DEXAMINEE :  FILE OF EXAMEINFO;
    DFDIR : FILE OF INDEX;
    DFTESTS : FILE OF SUBTEST;
    DPINFO : FILE OF PINFOREC;



BEGIN  (* end of day *)
  PAGE(OUTPUT);
  WRITELN;
  WRITE('Erase old contents of backup file ?  (Press ''Y'' or ''N'')  : ');
  IF GETCHAR(['y','Y','n','N'],TRUE,TRUE,TRUE) IN ['N','n'] THEN
    EXIT(ENDOFDAY);

  FOR I := 0 TO MAXEXAMINEE DO
    DIR[I].UNUSED := TRUE;
  RESET(FILEDIR,DONEINDEX);
  SEEK(FILEDIR,0);
  FILEDIR^ := DIR;
  PUT(FILEDIR);
  CLOSE(FILEDIR,LOCK);

  WRITELN;
  WRITELN;
  WRITELN('Backing up examinees who have finished session');
  WRITELN;
  LOADINDEX;
  COUNT := 0;
  FOR I := 0 TO MAXEXAMINEE DO
  BEGIN {1}
    IF NOT (DIR[I].UNUSED)  THEN
    BEGIN {2}
      LOADEXAMINEE(I);
      LOADPDATA(I);
      STATUSCODE := EXAMINEE.LASTTEST;
      IF STATUSCODE > GMAXSUBTEST THEN
      BEGIN {3}

        WRITELN;
        WRITE('Backing up ',EXAMINEE.ID);

        (* update the done directory *)
        WRITE('.');

        RESET(DFDIR,DONEINDEX);
        SEEK(DFDIR,0);
        DFDIR^[COUNT].UNUSED := FALSE;
        DFDIR^[COUNT].ID := DIR[I].ID;
        PUT(DFDIR);
        CLOSE(DFDIR,LOCK);

        (* update the done test taking  info *)

        WRITE('.');
        RESET(DEXAMINEE,DONEINFO);
        SEEK(DEXAMINEE,COUNT);
        DEXAMINEE^ := EXAMINEE;
        PUT(DEXAMINEE);
        CLOSE(DEXAMINEE,LOCK);

        (* save the personal data *)
        WRITE('.');

        RESET(DPINFO,DONEPINFO);
        SEEK(DPINFO,COUNT);
        DPINFO^ := PINFO;
        PUT(DPINFO);
        CLOSE(DPINFO,LOCK);

        (* update the done subtest results *)

        J := COUNT * GMAXSUBTEST;
        K := I * GMAXSUBTEST;
        REPEAT
          WRITE('.');
          LOADRESULTS(K);
          RESET(DFTESTS,DONERESULTS);
          SEEK(DFTESTS,J);
          DFTESTS^ := TESTS;
          PUT(DFTESTS);
          CLOSE(DFTESTS,LOCK);
          K := K + 1;
          J := J + 1;
        UNTIL J >= (COUNT * GMAXSUBTEST + 10);

        REMOVEEXAMINEE(I);
        COUNT := COUNT + 1;
      END;  {3}
    END;  {2}
  END;  {1}
  WRITELN;
  WRITELN;
  WRITELN(COUNT,' examinees backed up.');
  WRITELN;
  STALL;
END;   (* endofday  *)
```

```
(*****************************************************************************)
(*                                                                           *)
(*      Textfile : EMGR.DIR/E.SUMMARY.TEXT       Volume : TFILES             *)
(*      Codefile : E.MGR.CODE ('Include' file)   Volume : CATDATA            *)
(*                                                                           *)
(*****************************************************************************)
(* File last modified : August 5, 1983          NPROC                        *)
(*****************************************************************************)

(* list test scores to file/printer *)
PROCEDURE ESUMMARY;

CONST NONE = 0;
      B102222 = 1;
      B54321 = 2;
      B108642 = 3;
      TIMED = 4;

VAR TESTCOUNT,
    K,
    TINDEX,
    SLOT,
    CURRSTRAT,
    RSLOT,
    MINUTES,
    SECONDS : INTEGER;

    PRINTER,
    SCREEN,
    TOFILE : BOOLEAN;

    OPTION : CHAR;

    TNAME,
    ENAME,
    FNAME : STRING;

    SUMTIME,
    SAVETIME : REAL;

    {.........................................................................}

    (* list test scores to file *)
    PROCEDURE FILEENDSUMMARY(ESLOT : INTEGER);
    VAR MAXLINES,
        DUM,
        LINESOUT : INTEGER;

        {.....................................................................}

        (* display examinee personal data *)
        PROCEDURE SHOWEXAMINEE;
        VAR J : INTEGER;

            {.................................................................}

            (* display some data *)
            PROCEDURE D1;
            BEGIN
              WITH PINFO DO
              BEGIN {1}
                WRITELN(DEST);
                WRITE(DEST,LASTNAME,' ');
                WRITE(DEST,FIRSTNAME,' ');
                WRITE(DEST,MINITIAL);
                WRITE(DEST,EXAMINEE.ID);
                WRITE(DEST,CURRADDRESS);
                WRITE(DEST,HOMEOFREC);
                WRITE(DEST,CITIZENSHIP);
                WRITE(DEST,SEX);
                WRITELN(DEST,POPGROUP);
                      (* end of first line of compacted data *)
                WRITE(DEST,ETHNIC);
        WRITE(DEST,MARITAL);
        WRITE(DEST,DEPENDANTS);
        WRITE(DEST,BIRTHDATE);
      END;  {1}
    END;  (* d1 *)

 {.................................................................}

BEGIN  (* show examinee *)
  D1;
  WITH PINFO DO
  BEGIN {1}
    WRITE(DEST,EDUCATION);
    WRITE(DEST,TESTID);
    WRITE(DEST,AFQT);
    WRITELN(DEST,ASVAB);
        (* end of second line of compact data *)
    WRITE(DEST,ENLISTDATE);
    WRITE(DEST,ACTSERDATE);
    WRITE(DEST,ENL);
    WRITELN(DEST,AFEES);
        (* end of third line of compact data *)
    WRITE(DEST,POSTASVAB);
    WRITE(DEST,BOS);
    WRITE(DEST,STESTORDER[0],STESTORDER[1]);
    WRITELN(DEST,EXAMINEE.DATE);
  END;  {1}
END;  (* showexaminee *)

 {.................................................................}

(* check whether coding speed or numerical ops subtest needed to see  *)
(* if time info will appear at correct line for SPSS *)
PROCEDURE CHECKTIMEOUT;
BEGIN
  (* Test number minus 1.  ie:
      5. Numerical Operations       12. Numerical Operations III
      6. Coding Speed               13. Numerical Operations IV
     10. Coding Speed II            14. Coding Speed III
     11. Numerical Operations II    15. Coding Speed IV  *)
  IF EXAMINEE.TESTORDER[TINDEX] IN [5,6,10,11,12,13,14,15] THEN {a}
  BEGIN {1}
    IF EXAMINEE.TESTORDER[TINDEX] IN [6,10,14,15]
      THEN {b}
        MAXLINES := 7 * EXAMINEE.TESTLENGTH[TINDEX]
      ELSE {b}
        MAXLINES := 3 * EXAMINEE.TESTLENGTH[TINDEX]-1;
    IF LINESOUT < MAXLINES THEN {c}
      BEGIN {2}
        FOR DUM := LINESOUT+1 TO MAXLINES DO
          IF EXAMINEE.TESTORDER[TINDEX] IN [6,10,14,15] THEN {d}
            IF DUM MOD 7 = 0 THEN {e}
              BEGIN {3}
                WRITE(DEST,'Time000 ');
                WRITELN(DEST,SAVETIME : 6 : 1);
                SAVETIME := 0.0;
              END   {3}
              ELSE {e}
                WRITE(DEST,'Time000 ')
            ELSE {d}
            BEGIN {4}
              IF DUM MOD 3 = 0 THEN {f}
                BEGIN {5}
                  WRITE(DEST,'Time000 ');
                  WRITELN(DEST,SAVETIME : 6 : 1);
                  SAVETIME := 0.0;
                END   {5}
                ELSE {f}
                  WRITE(DEST,'Time000 ');
                IF DUM=MAXLINES THEN {g}
                  BEGIN {6}
                    WRITELN(DEST,'         ',SAVETIME : 6 : 1);
                    SAVETIME := 0.0;
```

```
         END;  {6}
      END;  {4}
   END   {2}
   ELSE {c}
     IF NOT (EXAMINEE.TESTORDER[TINDEX] IN [6,10,14,15])
      THEN {h}
        WRITELN(DEST,'        ',SAVETIME : 6 : 1);

 END;  {1}
END;   (* checktimeout *)

{------------------------------------------------------------------------}

PROCEDURE TIME;
BEGIN
 MINUTES := TESTS.STTIME DIV 60;
 SECONDS := TESTS.STTIME MOD 60;

 IF MINUTES > 0
  THEN {a}
    WRITE(DEST,MINUTES:3,':')
  ELSE {a}
    WRITE(DEST,'   :');

 IF SECONDS < 10
  THEN {b}
    WRITE(DEST,'0',SECONDS)
  ELSE {b}
    WRITE(DEST,SECONDS:2);

 MINUTES := TESTS.STINSTRTIME DIV 60;
 SECONDS := TESTS.STINSTRTIME MOD 60;

 IF MINUTES > 0
  THEN {c}
    WRITE(DEST,MINUTES:3,':')
  ELSE {c}
    WRITE(DEST,'   :');

 IF SECONDS < 10
  THEN {d}
    WRITE(DEST,'0',SECONDS)
  ELSE {d}
    WRITE(DEST,SECONDS:2);

 WRITE(DEST,TESTS.STPROCTCALLS:2);
   (*  line 6 of compact data *)
END;(*TIME*)

{------------------------------------------------------------------------}

(* send detailed or simple feedback to printer or screen *)
PROCEDURE OUTPUTRESULTS;
VAR J,I,K,TNUM : INTEGER;
  SEVEN : PACKED ARRAY[1..7] OF CHAR;

  {......................................................................}

  (* get info and write header *)
  PROCEDURE INFOHEADER;
  VAR SHORT_NAME : STRING;
  BEGIN
   RESET(FILEDIRECTORY,INDEXNAME);
   SEEK(FILEDIRECTORY,EXAMINEE.TESTORDER[TINDEX]);
   GET(FILEDIRECTORY);
   TNAME := FILEDIRECTORY^.TESTNAME;
   CLOSE(FILEDIRECTORY,LOCK);
   CURRSTRAT  := EXAMINEE.STRATEGY[TINDEX];
   SHORT_NAME := 'WHAT';  (* SET A DEFAULT *)

   CASE EXAMINEE.TESTORDER[TINDEX] OF
    0  :  SHORT_NAME := '  WK';
    1  :  SHORT_NAME := '  GS';
```

```
      2  :  SHORT_NAME := '  AR';
      3  :  SHORT_NAME := '  MK';
      4  :  SHORT_NAME := '  PC';
      5  :  SHORT_NAME := 'NO-1';
      6  :  SHORT_NAME := 'CS-1';
      7  :  SHORT_NAME := '  AI';
      8  :  SHORT_NAME := '  MC';
      9  :  SHORT_NAME := '  EI';
     10  :  SHORT_NAME := 'CS-2';
     11  :  SHORT_NAME := 'NO-2';
     12  :  SHORT_NAME := 'NO-3';
     13  :  SHORT_NAME := 'NO-4';
     14  :  SHORT_NAME := 'CS-3';
     15  :  SHORT_NAME := 'CS-4';
     16  :  SHORT_NAME := 'MK-2';
     17  :  SHORT_NAME := '  SI';
     18  :  SHORT_NAME := 'MC-2';
     19  :  SHORT_NAME := 'AR-2';
     20  :  SHORT_NAME := 'EI-2';
    END; (* cases *)

    WRITELN(DEST,SHORT_NAME);    (* end of line four of compact data *)
  END;  (* infoheader *)

  {-----------------------------------------------------------------}

  PROCEDURE CTOUT;
  BEGIN
    CASE CURRSTRAT OF
      NONE : WRITELN(DEST,'****************');
      B102222,
      B54321,
      B108642     : BEGIN {1}
                       WRITE(DEST,TESTS.ITEMINFO[I].THETA : 2 : 3);
                       WRITELN(DEST,TESTS.ITEMINFO[I].ERROR : 2 : 3);
                    END;  {1}
      TIMED       : BEGIN {2}
                       IF TESTS.ITEMINFO[I].RTYPE <> SEVENCHR THEN {a}
                       BEGIN {3}
                         IF ((I+1) > QUESTIONS) OR
                            ((I+1) > TESTS,NUMITEMS)
                           THEN {b}
                               CHECKTIMEOUT
                           ELSE {b}
                           BEGIN {4}
                            IF TESTS.ITEMINFO[I+1].ITEMNUM <= 0
                              THEN {c}
                                  CHECKTIMEOUT
                              ELSE {c}
                                  WRITELN(DEST,
                                     TESTS.ITEMINFO[I].LATENCY : 6 : 1);
                           END;  {4}
                       END;  {3}
                     END;  {2}
    END;  (* cases *)
  END;   (* ctout *)

{...................................................................}

BEGIN  (* outputresults *)

  INFOHEADER;
  I := 0;
  SUMTIME := 0.0;
  LINESOUT:=0;
  WHILE (I <= QUESTIONS) AND (I <= TESTS.NUMITEMS) DO
  BEGIN {1}
    IF TESTS.ITEMINFO[I].ITEMNUM > 0 THEN {a}
    BEGIN {2}
      IF TESTS.ITEMINFO[I].RTYPE <> SEVENCHR THEN {b}
        BEGIN {3}
          IF TESTS.ITEMINFO[I].ITEMNUM < 10
            THEN {c}
```

```
           WRITE(DEST,'000')
      ELSE {c}
          IF TESTS.ITEMINFO[I].ITEMNUM < 100
            THEN {d}
                WRITE(DEST,'00')
            ELSE {d}
                IF TESTS.ITEMINFO[I].ITEMNUM < 1000
                  THEN {e}
                      WRITE(DEST,'0');

    WRITE(DEST,TESTS.ITEMINFO[I].ITEMNUM,'*');
  END;  {3}

SAVETIME := TESTS.ITEMINFO[I].LATENCY;
IF CURRSTRAT = TIMED THEN
  SUMTIME := SUMTIME + SAVETIME;

CASE TESTS.ITEMINFO[I].RTYPE OF
  CHARVALUE : BEGIN {4}
                CASE TESTS.ITEMINFO[I].RESPONSE OF
                  'A'  :  WRITE(DEST,'1');
                  'B'  :  WRITE(DEST,'2');
                  'C'  :  WRITE(DEST,'3');
                  'D'  :  WRITE(DEST,'4');
                  'E'  :  WRITE(DEST,'5');
                END;  (* cases *)
                IF TESTS.ITEMINFO[I].CORRECT
                  THEN {f}
                      WRITE(DEST,'1')
                  ELSE {f}
                      WRITE(DEST,'0');
              END;  {4}
  INTVALUE  : BEGIN {5}
                WRITE(DEST,TESTS.ITEMINFO[I].INTRESPONSE : 1);
                IF TESTS.ITEMINFO[I].CORRECT
                  THEN {g}
                      WRITE(DEST,'1')
                  ELSE {g}
                      WRITE(DEST,'0');
              END;  {5}
  SEVENCHR  : BEGIN {6}
                J := TESTS.ITEMINFO[I].ACOUNT;
                FOR K := 1 TO J DO
                BEGIN {7}
                  IF TESTS.ITEMINFO[I].ITEMNUM < 10
                    THEN {h}
                        WRITE(DEST,'000')
                    ELSE {h}
                        IF TESTS.ITEMINFO[I].ITEMNUM < 100
                          THEN {i}
                              WRITE(DEST,'00')
                          ELSE {i}
                              IF TESTS.ITEMINFO[I].ITEMNUM < 1000
                                THEN {j}
                                    WRITE(DEST,'0');
                  WRITE(DEST,TESTS.ITEMINFO[I].ITEMNUM,K);

                  CASE TESTS.ITEMINFO[I].CHRRESPONSE[K] OF
                    'A'  :  WRITE(DEST,'1');
                    'B'  :  WRITE(DEST,'2');
                    'C'  :  WRITE(DEST,'3');
                    'D'  :  WRITE(DEST,'4');
                    'E'  :  WRITE(DEST,'5');
                  END; (* cases *)

                  IF TESTS.ITEMINFO[I].ACORRECT[K-1]
                    THEN {k}
                        WRITE(DEST,'1')
                    ELSE {k}
                        WRITE(DEST,'0');

                  WRITE(DEST,'*');
                  LINESOUT:=LINESOUT+1;
```

cGS

```
                          END;  {7}

                          TNUM := EXAMINEE.TESTORDER[TINDEX];

                          IF ((I+1) > QUESTIONS) OR
                             ((I+1) > TESTS.NUMITEMS) THEN {l}
                          BEGIN {8}
                            IF ((J = 7) AND (TNUM = 10)) OR
                               ((J = 3) AND (TNUM = 11))  THEN {m}
                            BEGIN {9}
                              WRITELN(DEST,
                                      TESTS.ITEMINFO[I].LATENCY:6:1);

                              IF J = 7 THEN {n}
                                SAVETIME := 0.0;  (* 420.0 - SUMTIME; *)

                              IF J = 3 THEN {o}
                                SAVETIME := 0.0;  (* 180.0 - SUMTIME; *)
                            END;  {9}
                            CHECKTIMEOUT;
                          END   {8}
                          ELSE {l}

                          BEGIN {10}
                            IF TESTS.ITEMINFO[I+1].ITEMNUM <= 0 THEN {p}
                            BEGIN {11}
                              IF ((J = 7) AND (TNUM = 10)) OR
                                 ((J = 3) AND (TNUM = 11)) THEN {q}

                              BEGIN {12}
                                WRITELN(DEST,
                                        TESTS.ITEMINFO[I].LATENCY:6:1);
                                IF J = 7
                                  THEN {r}
                                      SAVETIME := 0.0;    {420.0 - SUMTIME;}
                                IF J = 3
                                  THEN {s}
                                      SAVETIME := 0.0;    {180.0 - SUMTIME;}
                              END;  {12}

                              CHECKTIMEOUT;
                            END   {11}
                            ELSE {p}
                              WRITELN(DEST,TESTS.ITEMINFO[I].LATENCY:6:1);
                          END;  {10}

                        END;  {6}
          END;  (* cases *)
          CTOUT;
        END;  {2}
        I := I + 1;     (* each I outputs a line 5 of compact data *)
      END;  {1}
      TIME;
      IF EXAMINEE.PREDASVAB[TINDEX] < 10.0
      THEN
        BEGIN
             WRITE(DEST,' ');           (* RIGHT JUSTIFY PASVAB OUTPUT *)
             WRITELN(DEST,EXAMINEE.PREDASVAB[TINDEX]:5:2);
        END
      ELSE
             WRITELN(DEST,EXAMINEE.PREDASVAB[TINDEX]:5:2);
         (*  end of line 6 of compact data *)

    END;   (* outputresults *)

{............................................................................}

BEGIN   (* file end summary *)

  ENAME := '          ';
  FOR K := 0 TO 8 DO
    ENAME[K+1] := EXAMINEE.ID[K];
  FNAME := CONCAT('E',ENAME);
  FNAME := CONCAT(FNAME,'.TEXT');
  FNAME := CONCAT('QTEXT : ',FNAME);

  REWRITE(DEST,FNAME);

  (* write personal data to file *)
  SHOWEXAMINEE;
  SLOT := ESLOT;
  RSLOT := SLOT * GMAXSUBTEST;
  FOR TINDEX := 1 TO GMAXSUBTEST DO
  BEGIN {1}
    WRITE('.');
    CURRSTRAT := EXAMINEE.STRATEGY[TINDEX];
    LOADRESULTS(RSLOT);
    IF TESTS.NUMITEMS > 0 THEN {a}
    BEGIN {2}
      WRITELN(DEST);
      OUTPUTRESULTS;
    END;  {2}
    RSLOT := RSLOT + 1;
  END;  {1}
  MINUTES := EXAMINEE.TOTTIMECONSOLE DIV 60;
  SECONDS := EXAMINEE.TOTTIMECONSOLE MOD 60;
  WRITE(DEST,MINUTES:3,':');
  IF SECONDS < 10
    THEN {b}
        WRITE(DEST,'0',SECONDS)
    ELSE {b}
        WRITE(DEST,SECONDS:2);
  MINUTES := EXAMINEE.ORIENTATIONTIME DIV 60;
  SECONDS := EXAMINEE.ORIENTATIONTIME MOD 60;
  WRITE(DEST,MINUTES:3,':');
  IF SECONDS < 10
    THEN {c}
        WRITE(DEST,'0',SECONDS)
    ELSE {c}
  WRITE(DEST,SECONDS:2);
  WRITELN(DEST,EXAMINEE.NUMPROC:2);
  WRITELN(DEST);
  CLOSE(DEST,LOCK);
END;  (* file end summary *)

{------------------------------------------------------------------------}

(* display examinee personal data *)
PROCEDURE SHOWEXAMINEE;
VAR J : INTEGER;

    {.....................................................................}

    (* display some data *)
    PROCEDURE D1;
    BEGIN
      WITH PINFO DO
      BEGIN {1}
        WRITELN(DEST,'                        EXAMINEE PERSONAL DATA');
        WRITE(DEST,'Last Name');
        WRITE(DEST,' : ',LASTNAME);
        WRITE(DEST,'    First Name');
        WRITE(DEST,' : ',FIRSTNAME);
        WRITE(DEST,'    Middle Initial');
        WRITELN(DEST,' : ',MINITIAL);
        WRITELN(DEST,'Social Security #  : ',EXAMINEE.ID);
        WRITE(DEST,'Current Address (State)');
        WRITE(DEST,' : ',CURRADDRESS);
        WRITE(DEST,'        Home of Record (State)');
        WRITELN(DEST,' : ',HOMEOFREC);
        WRITE(DEST,'Citizenship');
        WRITE(DEST,' : ',CITIZENSHIP);
        WRITE(DEST,'                   Sex');
        WRITELN(DEST,' : ',SEX);
        WRITE(DEST,'Population Group');
        WRITE(DEST,' : ',POPGROUP);
        WRITE(DEST,'             Ethnic Group');
        WRITELN(DEST,' : ',ETHNIC);
        WRITE(DEST,'Marital Status');
        WRITELN(DEST,' : ',MARITAL);
        WRITE(DEST,'Number of Dependents');
        WRITELN(DEST,' : ',DEPENDANTS);
        WRITE(DEST,'Date of Birth');
        WRITELN(DEST,' : ',BIRTHDATE);
      END;  {1}
    END;  (* d1 *)

{.......................................................................}

BEGIN  (* show examinee *)
  IF SCREEN
    THEN {a}
        PAGE(OUTPUT);
  D1;
  WITH PINFO DO
  BEGIN {1}
    WRITE(DEST,'Education');
    WRITELN(DEST,' : ',EDUCATION);
    WRITE(DEST,'Test Id');
    WRITELN(DEST,' : ',TESTID);
    WRITE(DEST,'AFQT');
    WRITELN(DEST,' : ',AFQT);
    WRITE(DEST,'ASVAB Scores');
    WRITELN(DEST,' : ',ASVAB);
    WRITE(DEST,'Date of Enlistment');
    WRITELN(DEST,' : ',ENLISTDATE);
    WRITE(DEST,'Active Service Date');
    WRITELN(DEST,' : ',ACTSERDATE);
    WRITE(DEST,'Rating / MOS');
    WRITELN(DEST,' : ',ENL);
    WRITE(DEST,'AFEES');
    WRITELN(DEST,' : ',AFEES);
    WRITE(DEST,'Branch of Service');
    WRITELN(DEST,' : ',BOS);
    WRITE(DEST,'Retest ASVAB');
    WRITELN(DEST,' : ',POSTASVAB);
    WRITE(DEST,'Test Order');
    WRITELN(DEST,' : ',STESTORDER[0],STESTORDER[1]);
  END;  {1}
END;  (* showexaminee *)

{-----------------------------------------------------------------------}

(* send detailed or simple feedback to printer or screen *)
PROCEDURE OUTPUTRESULTS;
VAR J,I,K : INTEGER;
    SEVEN : PACKED ARRAY[1..7] OF CHAR;

    {...................................................................}

    (* get info and write header *)
    PROCEDURE INFOHEADER;
    BEGIN
      IF SCREEN
        THEN {a}
            PAGE(OUTPUT);
      RESET(FILEDIRECTORY,TINDEXNAME);
      SEEK(FILEDIRECTORY,EXAMINEE.TESTORDER[TINDEX]);
      GET(FILEDIRECTORY);
      TNAME := FILEDIRECTORY^.TESTNAME;
      CLOSE(FILEDIRECTORY,LOCK);

      CURRSTRAT  := EXAMINEE.STRATEGY[TINDEX];

      WRITELN(DEST,' ':25,'Test : ',TNAME);
      WRITELN(DEST);
      WRITE(DEST,'Item #    Response(s)  Correct     Ability     Variance');
      WRITELN(DEST,'  Elapsed Time (sec)');
      WRITE(DEST,'------    ----------   -------     -------     --------');
    WRITELN(DEST,'   ------------------');

  END;  (* infoheader *)

  {-----------------------------------------------------------------------}

  (*  display time information *)
  PROCEDURE TIMEINFO;
  BEGIN
    WRITELN(DEST);
    MINUTES := TESTS.STTIME DIV 60;
    SECONDS := TESTS.STTIME MOD 60;
    WRITE(DEST,'Elapsed Time for subtest : ');
    IF MINUTES > 0
      THEN {a}
          WRITE(DEST,MINUTES,' minute(s) and ');
    WRITELN(DEST,SECONDS,' second(s).');
    MINUTES := TESTS.STINSTRTIME DIV 60;
    SECONDS := TESTS.STINSTRTIME MOD 60;
    WRITE(DEST,'Elapsed time for instructions : ');
    IF MINUTES > 0
      THEN {b}
          WRITE(DEST,MINUTES,' minute(s) and ');
    WRITELN(DEST,SECONDS,' second(s).');
    WRITELN(DEST,'# of Subtest proctor calls : ',TESTS.STPROCTCALLS);
    IF SCREEN THEN {c}
      BEGIN {1}
        WRITELN;
        STALL;
      END;  {1}
  END;    (* timeinfo *)

{.......................................................................}

BEGIN  (* outputresults *)

  INFOHEADER;
  I := 0;
  WHILE (I <= QUESTIONS) AND (I <= TESTS.NUMITEMS) DO
  BEGIN {1}
    IF TESTS.ITEMINFO[I].ITEMNUM > 0 THEN {a}
    BEGIN {2}
      IF TESTS.ITEMINFO[I].RTYPE <> SEVENCHR
        THEN {b}
            WRITE(DEST,TESTS.ITEMINFO[I].ITEMNUM : 4);

      CASE TESTS.ITEMINFO[I].RTYPE OF
        CHARVALUE : BEGIN {3}
                      WRITE(DEST,TESTS.ITEMINFO[I].RESPONSE : 10);
                      IF TESTS.ITEMINFO[I].CORRECT
                        THEN {c}
                            WRITE(DEST,'            Yes  ')
                        ELSE {c}
                            WRITE(DEST,'            No   ');
                    END;  {3}
        INTVALUE  : BEGIN {4}
                      WRITE(DEST,TESTS.ITEMINFO[I].INTRESPONSE : 11);
                      IF TESTS.ITEMINFO[I].CORRECT
                        THEN {d}
                            WRITE(DEST,'            Yes  ')
                        ELSE {d}
                            WRITE(DEST,'            No   ');
                    END;  {4}
        SEVENCHR  : BEGIN {5}
                      J := TESTS.ITEMINFO[I].ACOUNT;
                      FOR K := 1 TO J DO
                      BEGIN {6}
                        WRITE(DEST,TESTS.ITEMINFO[I].ITEMNUM : 4);
                        WRITE(DEST,K,' ':10);
                        WRITE(DEST,TESTS.ITEMINFO[I].CHRRESPONSE[K]);
                        IF TESTS.ITEMINFO[I].ACORRECT[K-1]
                          THEN {e}
                              WRITE(DEST,'          Yes  ')
```

```
                         ELSE {e}
                             WRITE(DEST,'          No   ');
                       WRITE(DEST,'     *****       *****   ');
                       IF K = J
                         THEN {f}
                             WRITELN(DEST,TESTS.ITEMINFO[I].LATENCY : 9:1)
                         ELSE {f}
                             WRITELN(DEST,'     ***** ')
                     END;  {6}
                     WRITELN(DEST);
                   END;  {5}
      END;  (* cases *)

      CASE CURRSTRAT OF
        NONE : WRITELN(DEST,'     *****      *****     *****');
        B102222,
        B54321,
        B108642     : BEGIN {7}
                        WRITE(DEST,TESTS.ITEMINFO[I].THETA : 12 : 3);
                        WRITE(DEST,TESTS.ITEMINFO[I].ERROR : 12 : 3);
                        WRITELN(DEST,'        *****');
                      END;  {7}
        TIMED       : BEGIN {8}
                        IF TESTS.ITEMINFO[I].RTYPE <> SEVENCHR THEN {g}
                        BEGIN {9}
                          WRITE(DEST,'     *****       *****   ');
                          WRITELN(DEST,TESTS.ITEMINFO[I].LATENCY : 9 : 1);
                        END;  {9}
                      END;  {8}
      END;  (* cases *)
    END;  {2}
    I := I + 1;
  END;  {1}
  TIMEINFO;
END;    (* outputresults *)

{----------------------------------------------------------------------}

(*  get which examinee and where to send summary *)
PROCEDURE WHOANDWHERE;
BEGIN
  LOADINDEX;
  ENTERID;
  SLOT := DIRINDEXNUM(EXAMINEE.ID);

  IF SLOT < 0 THEN {a}
    BEGIN {1}
      GOTOXY(0,12);
      WRITELN('No record in file with ID ',EXAMINEE.ID);
      WRITELN;
      STALL;
      EXIT(ESUMMARY);
    END   {1}

    ELSE {a}

    BEGIN {2}
      PAGE(OUTPUT);
      GOTOXY(18,0);
      WRITE('OUTPUT SELECT MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following options by entering its number.');
      GOTOXY(16,9);
      WRITE('1.  QUIT');
      GOTOXY(16,10);
      WRITE('2.  SUMMARY TO CONSOLE');
      GOTOXY(16,11);
      WRITE('3.  SUMMARY TO PRINTER');
      GOTOXY(16,12);
      WRITE('4.  SUMMARY TO FILE');
      GOTOXY(16,16);
      WRITE('Enter Choice # : ');
      OPTION :=  GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
```

```
          TOFILE := FALSE;
          SCREEN := FALSE;
          PRINTER := FALSE;

          CASE OPTION OF
            '1' : EXIT(ESUMMARY);
            '2' : BEGIN {3}
                    REWRITE(DEST,'CONSOLE : ');
                    SCREEN := TRUE;
                  END;  {3}
            '3' : BEGIN {4}
                    PRINTER := TRUE;
                    REWRITE(DEST,UNITNUMPRINTER);
                  END;  {4}
            '4' : BEGIN {5}
                    TOFILE := TRUE;
                    PAGE(OUTPUT);
                    WRITE('Writing examinee records to file .');
                    LOADEXAMINEE(SLOT);
                    LOADPDATA(SLOT);
                    FILEENDSUMMARY(SLOT);
                  END;  {5}
          END;  (* cases *)
        END;  {2}
    END;(* who and where *)

{..............................................................................}

BEGIN   (* examinee summary *)

    (*  get which examinee *)
    WHOANDWHERE;

    IF NOT TOFILE THEN {a}
    BEGIN {1}

      (*  load examinee testtaking info *)
      LOADEXAMINEE(SLOT);

      (* load examinee personal data *)
      LOADPDATA(SLOT);

      SHOWEXAMINEE;
      IF SCREEN THEN {b}
        BEGIN {2}
          WRITELN(DEST);
          STALL;
          PAGE(OUTPUT);
        END  {2}
        ELSE {b}
            FOR K := 1 TO 5 DO
              WRITELN(DEST);
      RSLOT := SLOT * GMAXSUBTEST;
      TESTCOUNT := 0;
      IF EXAMINEE.LASTTEST <> GMAXSUBTEST THEN {c}
      BEGIN {3}
        FOR TINDEX := 1 TO GMAXSUBTEST DO
        BEGIN {4}
          CURRSTRAT := EXAMINEE.STRATEGY[TINDEX];
          LOADRESULTS(RSLOT);
          IF TESTS.NUMITEMS > 0 THEN {d}
          BEGIN {5}
            TESTCOUNT := TESTCOUNT + 1;
            IF NOT SCREEN THEN {e}
            BEGIN {6}
              FOR K := 1 TO 80 DO
                WRITE(DEST,'*');
              WRITELN(DEST);
              WRITELN(DEST);
            END;  {6}
            OUTPUTRESULTS;
            FOR K := 1 TO 2 DO WRITELN(DEST);
          END;  {5}
          RSLOT := RSLOT + 1;
        END;  {4}
      END;  {3}
      IF SCREEN
        THEN {f}
            PAGE(OUTPUT)
        ELSE {f}
        BEGIN {7}
          FOR K := 1 TO 80 DO
            WRITE(DEST,'*');
          WRITELN(DEST);
          WRITELN(DEST);
        END;  {7}
      IF TESTCOUNT <> 0 THEN {g}
      BEGIN {8}
        MINUTES := EXAMINEE.TOTTIMECONSOLE DIV 60;
        SECONDS := EXAMINEE.TOTTIMECONSOLE MOD 60;
        WRITELN(DEST,'Total session elapsed time : ',MINUTES,' minute(s) and ',
                SECONDS,' second(s).');
        MINUTES := EXAMINEE.ORIENTATIONTIME DIV 60;
        SECONDS := EXAMINEE.ORIENTATIONTIME MOD 60;
        WRITELN(DEST,'Familiarization elapsed time : ',MINUTES,' minute(s) and ',
                SECONDS,' second(s).');
        WRITELN(DEST,'# session proctor calls : ',EXAMINEE.NUMPROC);
        WRITELN(DEST);
      END   {8}
      ELSE {g}
      BEGIN {9}
        WRITELN(DEST);
        WRITELN(DEST);
        WRITELN(DEST,'Examinee has not taken any subtests.');
        WRITELN(DEST);
      END;  {9}
      IF SCREEN
        THEN {h}
            STALL;
      CLOSE(DEST,LOCK);
    END;  {1}

    IF TOFILE THEN {i}
    BEGIN {10}
      WRITELN;
      WRITELN;
      WRITE('Summary of examinee is in file  ',FNAME);
      WRITELN;
      WRITELN;
      STALL;
    END;  {10}
END;  {* esummary *}
```

```
(***************************************************************************)
(*                                                                         *)
(*      Textfile : EMGR.DIR/E.ZERO.TEXT          Volume : TFILES           *)
(*      Codefile : E.MGR.CODE ('Include' file)   Volume : CATDATA          *)
(*                                                                         *)
(***************************************************************************)
(*              DEC.  1, 1982                    NPROC                     *)
(***************************************************************************)

(**** initialize the directory of a file ****)
PROCEDURE ZERODIRECTORY;
VAR MAXRECORDS,
    I : INTEGER;

    {.......................................................................}

    (* zero out the examinee directory *)
    PROCEDURE ZAPINDEX;
    BEGIN
      FOR I := 0 TO MAXEXAMINEE DO
        DIR[I].UNUSED := TRUE;
      RESET(FILEDIR,INDEXNAME);
      SEEK(FILEDIR,0);
      FILEDIR^ := DIR;
      PUT(FILEDIR);
      CLOSE(FILEDIR,LOCK);
    END;   (* zap index *)

    {-----------------------------------------------------------------------}

    (* zero out the examinee test taking information *)
    PROCEDURE ZAPTESTINFO;
    BEGIN
      RESET(FILEEXAMINEE,INFONAME);
      SEEK(FILEEXAMINEE,0);
      FOR I := 0 TO MAXEXAMINEE DO
      BEGIN {1}
        FILEEXAMINEE^ := EXAMINEE;
        PUT(FILEEXAMINEE);
      END;  {1}
      CLOSE(FILEEXAMINEE,LOCK);
    END;  (* zaptestinfo *)

    {-----------------------------------------------------------------------}

    (* zap the personal info of examinee *)
    PROCEDURE ZAPPERSONALINFO;
    BEGIN

      RESET(PINFOFILE,PINFONAME);
      SEEK(PINFOFILE,0);
      FOR I := 0 TO MAXEXAMINEE DO
      BEGIN {1}
        PINFOFILE^ := PINFO;
        PUT(PINFOFILE);
      END;  {1}
      CLOSE(PINFOFILE,LOCK);
    END;   (*  zappersonalinfo  *)

    {-----------------------------------------------------------------------}

    (*  zap examinee test results *)
    PROCEDURE ZAPRESULTS;
    VAR J,K : INTEGER;
    BEGIN
      K := (MAXEXAMINEES * STESTS) + STESTS;
      RESET(FILETESTS,RESULTS);
      SEEK(FILETESTS,0);
      WRITELN;
      FOR I := 0 TO K DO
      BEGIN {1}
        IF (I MOD 5) = 0 THEN WRITE('.');
        FILETESTS^ := TESTS;
        PUT(FILETESTS);
      END;  {1}
      CLOSE(FILETESTS,LOCK);
    END;   (*  zapresults  *)

 {...........................................................................}

BEGIN (* zero directory *)
  PAGE(OUTPUT);
  SQUAWK;
  GOTOXY(16,2);
  WRITE('***********     WARNING    ***********');
  GOTOXY(0,5);
  WRITELN(
'You have selected the purge directory option.  This will flush away all');
  WRITELN('existing examinee data in the files listed below.');
  GOTOXY(7,9);
  WRITE(INDEXNAME);
  GOTOXY(7,10);
  WRITE(INFONAME);
  GOTOXY(7,11);
  WRITE(RESULTS);
  GOTOXY(7,12);
  WRITE(PINFONAME);
  GOTOXY(0,14);
  IF BACKUP THEN
    WRITE('These are the DONE EXAMINEE files.')
  ELSE
    WRITE('These are the SESSION files.');
  WRITELN;
  WRITE('Do you wish to purge these files?  (Press  ''N'' or ''Y'') : ');
  IF GETCHAR(['y','n','Y','N'],TRUE,TRUE,TRUE) IN ['Y','y'] THEN
  BEGIN {1}
    WRITELN;
    WRITELN;
    WRITELN('Last chance !!!!!!!!!');
    SQUAWK;
    WRITE('Are you sure you want to purge ?  (Press ''N'' or ''Y'') : ');
    IF GETCHAR(['y','n','Y','N'],TRUE,TRUE,TRUE) IN ['Y','y']  THEN
    BEGIN {2}
      PAGE(OUTPUT);
      WRITE('Purging files.');
      INITEREC;
      ZAPINDEX;
      ZAPTESTINFO;
      ZAPPERSONALINFO;
      ZAPRESULTS;

    END;  {2}
  END;  {1}
END;  (* zero directory *)
```

SMGR.DIR:
Subdirectory - Strategy Manager Textfiles

```
(***************************************************************************)
(*                                                                         *)
(*      Textfile : SMGR.DIR/S.MGR.TEXT          Volume : TFILES            *)
(*      Codefile : S.MGR.CODE TRAT              Volume : CATDATA           *)
(*                                                                         *)
(***************************************************************************)
(*  File last modified : Feb 23, 1983           NPRDC                      *)
(***************************************************************************)
(*     This program allows access to the various data structures used by the *)
(* different strategies of computer adaptive testing methods.  Through this  *)
(* access, one can see how the strategies are set up, change data in the     *)
(* files, or enter new data.  Currently, only infotables for the Bayesian    *)
(* method are implemented.                                                   *)
(***************************************************************************)

(*$S+*)
PROGRAM STRATEGY;
USES CHAINSTUFF,
     REALMODES,
     TRANSCEND;

CONST (* ascii values *)
      ETX = 3;
      BELL = 7;
      NUL = 0;
      LARROW = 8;
      RARROW = 21;
      RET = 13;
      UP = 11;
      DOWN = 10;
      ESC = 27;
      SPACE = 32;
      NIL = -1;
      ASCIIOFFSET = 48; (* ascii zero *)
      MAXLINEBUF = 79;   (* string buffer size *)

      (* infotable size *)
      TMIN = -2.25;   (* Lowest value of ability level, Theta *)
      DT = 0.125;   (* Increment value -- Theta in each row in table is DT
                                         larger than previous row *)
      (* test directory name *)
      INDEXNAME = 'CATDATA:TESTINDEX.DATA'; (* test directory *)
      DATANAME = 'CATDATA:ITEMPOOL.DATA';  (* Question directory *)

      (* slots available in directory *)
      MAXSUBTESTS = 20;

      (* maximum question pool per test *)
      MAXITEMPOOL = 300;

      (* maximum # of sample questions *)
      MAXSAMPLES = 5;

      UNITNUMPRINTER = 'PRINTER:';

      DEFAULTFILE = 'STRAT-INFO.TEXT';

      VERSION = '[1.03]';

TYPE DIRDATA = PACKED RECORD  (* directory for tests *)
                    UNUSED : BOOLEAN;
                    TESTNAME : STRING;
                    ITEMCODE : PACKED ARRAY
                               [0..MAXITEMPOOL]
                               OF INTEGER;
                  END;

     (* type of question response *)
     SEVENTYPE = PACKED ARRAY[1..7] OF CHAR;

     (* Different types of ways to answer a question *)
     ITEMRESPONSES = (CHARVALUE,      (* normal multiple choice *)
                      INTVALUE,       (* Integer value as answer *)
                      SEVENCHR);       (* seven characters saved as answer *)

     (* question ptrs/data , information for each question *)
     ITEMDATA = PACKED RECORD

                         (* flags if graphics item *)
                         GRAPHICS : BOOLEAN;

                         (* valid response ranges for multiple choice *)
                         LOWANSWER,
                         HIGHANSWER : CHAR;

                         (* block # in file where text starts *)
                         ITEMBLOCK,

                         (* byte # in block where text starts *)
                         ITEMPTR,

                         (* # of answers if multiple question screen *)
                         ANSWERCOUNT  : INTEGER;

                         (* information parameters for bayesian strategy *)
                         A,B,C,

                         (* currently unused *)
                         PROPCORRECT,
                         POINTBISERIAL,
                         YOPT,
                         XOPT,
                         DUMMY1,
                         DUMMY2,
                         DUMMY3 : REAL;

                         (* correct answer to question *)
                         CASE ATYPE : ITEMRESPONSES OF
                           CHARVALUE : (ANSWER : CHAR);
                           INTVALUE  : (INTANSWER : INTEGER);
                           SEVENCHR  : (CHRANSWER : SEVENTYPE);
                       END;

     SETOFCHAR = SET OF CHAR;

VAR LETTERS,DIGITS,CHARACTERS : SET OF CHAR;
    output,
    COMMAND : CHAR;

    ESCPROC : BOOLEAN;

    (* string character buffer *)
    LINEBUF : PACKED ARRAY[0..MAXLINEBUF] OF CHAR;

    CURRINDEXRECNUM : INTEGER;  (* record # of file directory *)

    (* test directory *)
    DIRECTORY : DIRDATA;
    FILEDIRECTORY : FILE OF DIRDATA;

    (* test question ptrs/data *)
    ITEMINFO : ITEMDATA;
    FILEITEMINFO : FILE OF ITEMDATA;

    (* output file for test listings *)
    DEST : TEXT;

    (* directory record information *)
    DIRINFO : ARRAY[0..MAXSUBTESTS] OF RECORD

                                        (* record occupied *)
                                        NOTUSED : BOOLEAN;

                                        (* subtest name *)
                                        TNAME : STRING;
```

```
                                (* # items in subtest *)
                                ITEMCOUNT : INTEGER;
                              END;

(* ..................................................................*)

(* This procedure is called by most of the other procedures.  It clears *)
(* the console screen. *)
PROCEDURE PAGE(DUMMY : CHAR);
BEGIN
  WRITE(CHR(28));
  GOTOXY(0,0);
END;

(* --------------------------------------------------------------------- *)

(*$I /TFILES/SMGR.DIR/S.UTL.TEXT *)      (* utility procedures *)

(* --------------------------------------------------------------------- *)

(* This procedure writes the directory information to a record.          *)
(* It puts into an array the names of the subtests.  If a test had been *)
(* deleted from the file, there will be no name in the array location   *)
(* corresponding to the place in the file where it had been, and a      *)
(* boolean is set to indicate that it is blank.                          *)
(* This procedure is called by: Program STRATEGY main routine *)

PROCEDURE GETDIRINFO;
VAR I,K,ICOUNT : INTEGER;
    (* all other variables are global to Program STRATEGY *)

BEGIN  (* get dir info *)

  (* initialize the directory information *)
  FOR I := 0 TO MAXSUBTESTS DO
    DIRINFO[I].NOTUSED := TRUE;

  (* get the directory information *)
  I := 0;
  RESET(FILEDIRECTORY,INDEXNAME);
  REPEAT
    SEEK(FILEDIRECTORY,I);
    GET(FILEDIRECTORY);

    IF NOT (FILEDIRECTORY^.UNUSED)
      THEN
        BEGIN {1}
          DIRECTORY := FILEDIRECTORY^;
          DIRINFO[I].NOTUSED := FALSE;
          DIRINFO[I].TNAME := DIRECTORY.TESTNAME;
          ICOUNT := 0;
          DIRINFO[I].ITEMCOUNT := ICOUNT;
        END;  {1}

    I := I + 1;
  UNTIL I > MAXSUBTESTS;
  CLOSE(FILEDIRECTORY,NORMAL);
END;  (* getdirinfo *)

(* --------------------------------------------------------------------- *)

(* Given a question code, this function returns the location *)
(* of the question's data, & text pointers.                  *)
(* This function is called by: Procedure ? *)

FUNCTION SLOT(CODE : INTEGER) : INTEGER;
VAR INDEX : INTEGER;
    FOUND : BOOLEAN;
    (* All other variables are global to Program STRATEGY *)

BEGIN  (* slot *)
  INDEX := MAXSAMPLES + 1;
  FOUND := FALSE;
  REPEAT
    IF DIRECTORY.ITEMCODE[INDEX] = CODE
      THEN
          FOUND := TRUE
      ELSE
          INDEX := INDEX + 1;
  UNTIL (INDEX > MAXITEMPOOL) OR (FOUND);

  IF FOUND
    THEN
        SLOT := INDEX
    ELSE
        SLOT := NIL;      (* nil here is not a pointer, but an integer *)
END;  (* slot *)

(* ------------------------------------------------------------------- *)

(* This procedure lists the names of the tests that are in the database, *)
(* and loads the record containing itemcode numbers for the single test  *)
(* selected. This procedure is called by the following procedures :      *)
(* NEWINFO in textfile STRAT-NEW, LISTINFO in textfile STRAT-LIST,       *)
(* VERIFY in textfile STRAT-VERF, MODIFYINFO in texfile STRAT-MODF, and  *)
(* FINDINFO in textfile STRAT-FIND. *)

PROCEDURE LOADTEST(MESSAGE : STRING);
(* Constant MAXSUBTESTS is global to Program STRATEGY. *)
VAR Q,
    TESTNUM,
    RECNUM : INTEGER;
    OKTEST : BOOLEAN;
    TEXTCODE : CHAR;
    (* All other variables are global to Program STRATEGY. *)

    (* ............................................................. *)

    (* This procedure lists a directory of the test names to the screen. *)
    (* This procedure is called by: Procedure LOADTEST *)

    PROCEDURE LISTTESTS;
    (* Constant MAXSUBTESTS is global to Program STRATEGY. *)
    VAR I,J : INTEGER;
        (* DIRINFO.NOTUSED and .TNAME are global to Program STRATEGY. *)

    BEGIN  (* list tests *)
      PAGE(OUTPUT);
      GOTOXY(24,0);
      WRITE('LIST OF SUBTESTS');
      I := 0;
      J := 0;
      GOTOXY(0,3);

      REPEAT
        IF NOT (DIRINFO[I].NOTUSED)      (* The test exists *)
          THEN
              BEGIN {1}
                J := J + 1;
                IF J <= 10
                  THEN
                      GOTOXY(0,2+J)
                  ELSE
                      GOTOXY(40,2+J-10);
                WRITE(J,'.  ',DIRINFO[I].TNAME);
              END;  {1}
        I := I + 1;
      UNTIL I > MAXSUBTESTS;
    END;  (* listtests *)

(* ................................................................. *)

BEGIN  (* loadtest *)
  OKTEST := FALSE;
  LISTTESTS;
  RESET(FILEDIRECTORY,INDEXNAME);

  REPEAT
    GOTOXY(0,15);
    WRITELN('INSTRUCTIONS : Enter choice #, then press <RET>.');
    WRITE('               To escape, press 0 then <RET>.');
    GOTOXY(0,18);
    WRITE(MESSAGE);
  (*$I-*)
    READLN(TESTNUM);
  (*$I+*)

    IF TESTNUM = 0
      THEN
          BEGIN {1}
            ESCPROC := TRUE;
            CLOSE(FILEDIRECTORY,LOCK);
            EXIT(LOADTEST);
          END;  {1}

    IF (TESTNUM < 0) OR (TESTNUM >  (MAXSUBTESTS+1))
      THEN
          BEGIN {2}
            WRITELN;
            WRITELN('Invalid test # : ',TESTNUM);
            SQUAWK;     (* Procedure located in "include" file T-UTL *)
            WRITELN;
            STALL;      (* Procedure located in "include" file T-UTL *)
          END   {2}
      ELSE
          BEGIN {3}
            RECNUM := 0;
            Q := 0;

            REPEAT
              SEEK(FILEDIRECTORY,RECNUM);
              GET(FILEDIRECTORY);
              IF NOT (FILEDIRECTORY^.UNUSED) THEN Q := Q + 1;
              RECNUM := RECNUM + 1;
            UNTIL (Q = TESTNUM) OR (RECNUM > MAXSUBTESTS);

            IF Q = TESTNUM
              THEN
                  BEGIN {4}
                    CURRINDEXRECNUM := RECNUM - 1;
                    OKTEST := TRUE;
                  END   {4}
              ELSE
                  BEGIN {5}
                    WRITELN;
                    WRITELN('No test loaded');
                    WRITELN;
                    STALL; (* Procedure located in "include" file T-UTL *)
                  END;  {5}
           END;  {3}
    IF NOT OKTEST THEN BLANKLINES(18,6,18);
  UNTIL OKTEST;

  SEEK(FILEDIRECTORY,CURRINDEXRECNUM);
  GET(FILEDIRECTORY);
  DIRECTORY := FILEDIRECTORY^;
  CLOSE(FILEDIRECTORY,LOCK);
END;  (* Loadtest *)

(* ---------------------------------------------------------------------- *)

(* Show command level selections *)
(* This procedure is called by Program STRATEGY main routine. *)

PROCEDURE MENU;
BEGIN
  PAGE(OUTPUT);
  GOTOXY(20,0);
  WRITE('STRATEGY DATABASE MENU ',VERSION);
  GOTOXY(0,4);
  WRITE('Select one of the following options by entering its number.');
  GOTOXY(16,8);
  WRITE('1.  QUIT');
  GOTOXY(16,9);
  WRITE('2.  INFO TABLE MANAGEMENT');
  GOTOXY(16,10);
  WRITE('3.  PYRAMIDAL MANAGEMENT');
  GOTOXY(16,11);
  WRITE('4.  FLEXILEVEL MANAGEMENT');
  GOTOXY(16,15);
  WRITE('Enter Choice # : ');
END; (* menu *)

(* ------------------------------------------------------------------------ *)

(* This procedure is called by Program STRATEGY main routine. *)

PROCEDURE INFOSETUP;
CONST(* information tables *)
      TABNAME = 'CATDATA:TABINFO.DATA';

      (* output file for table listing *)
      DEFAULTFILE = 'CATDATA:INFO.RSLTS.TEXT';

      (* information table dimensions *)
      INFOROW = 36;
      INFOCOLUMN = 20;

TYPE TABLE = ARRAY[1..INFOCOLUMN,1..INFOROW] OF INTEGER;
     PARAMETER = PACKED RECORD
                          ITEM : INTEGER;
                          A,              (* Discrimination parameter *)
                          B,              (* Difficulty parameter *)
                          C : REAL;       (* Guessing coefficient *)
                         END;

      SORTTYPE = PACKED RECORD
                          ITEM : INTEGER;
                          INFOVALUE : REAL;
                        END;

VAR N,
    MINITEMPOOL : INTEGER;
    TESTPARAM : PACKED ARRAY[MAXSAMPLES..MAXITEMPOOL] OF PARAMETER;
    SORTARRAY : PACKED ARRAY[0..MAXITEMPOOL] OF SORTTYPE;
    COMMAND : CHAR;
    INFOTABLE : TABLE;
    INFOFILE  : FILE OF TABLE;
    OTHERINFO : BOOLEAN;

    (* .................................................................... *)

    (* This procedure updates the information table file for subtest. *)
    (* This procedure is called by: Procedure *)

    PROCEDURE UPDATEINFO(RECNUM : INTEGER);
    (* Constant TABNAME is defined in Procedure INFOSETUP. INFOTABLE and *)
    (* INFOFILE are declared in Procedure INFOSETUP. INFOTABLE is an     *)
    (* array whose values are determined in Procedure *)

    BEGIN  (* update info *)
      RESET(INFOFILE,TABNAME);
      SEEK(INFOFILE,RECNUM);
      INFOFILE^ := INFOTABLE;
      PUT(INFOFILE);
      CLOSE(INFOFILE,LOCK);
    END;  (* updateinfo *)

    (* -------------------------------------------------------------------- *)

    (* This procedure loads the information table for a subtest. *)
(* This procedure is called by: Procedure *)

PROCEDURE LOADINFO(RECNUM : INTEGER);
(* Constant TABNAME is defined in Procedure INFOSETUP. INFOTABLE and *)
(* INFOFILE are declared in Procedure INFOSETUP. INFOTABLE is an     *)
(* array.  *)

BEGIN
  RESET(INFOFILE,TABNAME);
  SEEK(INFOFILE,RECNUM);
  GET(INFOFILE);
  INFOTABLE := INFOFILE^;
  CLOSE(INFOFILE,NORMAL);
END;  (* loadinfo *)

(* ----------------------------------------------------------------- *)

(* This procedure formats a screen display to enter or modify *)
(* information table items manually. *)
(* This procedure is called by: Procedure MODIFYINFO *)

PROCEDURE FORMATSCREEN;
VAR I : INTEGER;
BEGIN
  PAGE(OUTPUT);
  GOTOXY(18,1);
  WRITELN('1',' ':18,'20');
  GOTOXY(17,2);
  WRITELN('_____________________');
  GOTOXY(15,3);
  WRITE('1');
  FOR I := 1 TO 17 DO
  BEGIN {1}
    GOTOXY(17,I+2);
    WRITELN('|                   |');
  END;  {1}
  GOTOXY(14,20);
  WRITELN('36 |___________________|');
  GOTOXY(0,5);
  WRITELN('Row : ');
  WRITELN;
  WRITELN('Column : ');
  WRITELN;
  WRITELN('Value   : ');
END;  (* format screen *)

(* ----------------------------------------------------------------- *)

(* This function is called by: Procedure Param_Array *)

FUNCTION HASH(KEY : INTEGER) : INTEGER;
BEGIN
  HASH := (CURRINDEXRECNUM * MAXITEMPOOL)
          + KEY + CURRINDEXRECNUM;
END; (* Hash *)

(* ----------------------------------------------------------------- *)

(* This procedure gets the a,b & c parameters of the desired subtest *)
(* from the file database. *)
(* This procedure is called by: Procedure Newinfo *)

PROCEDURE PARAM_ARRAY;
  (* Constants DATANAME, MAXITEMPOOL, & MAXSAMPLES are global to *)
  (* Program STRATEGY.  *)
VAR MINITEMPOOL,I : INTEGER;
  (* DIRECTORY, FILEITEMINFO & ITEMINFO are global to *)
  (* Program STRATEGY. TESTPARAM is declared in Procedure INFOSETUP *)
  (* DIRECTORY is set in Procedure LOADTEST. *)

BEGIN  (* param array *)
  (* Get all data items for this test *)
  WRITELN;
```

```
    WRITELN('LOADING PARAMETERS FROM FILE.');
    RESET(FILEITEMINFO,DATANAME);
    MINITEMPOOL:= MAXSAMPLES + 1;
    I:=HASH(MINITEMPOOL);
    SEEK(FILEITEMINFO,I);
    FOR I := MINITEMPOOL TO MAXITEMPOOL DO
      BEGIN (1)
        IF (I MOD 10) = 0 THEN WRITE('.');

        (* Put item, a, b, c into an array *)
        GET(FILEITEMINFO);
        ITEMINFO := FILEITEMINFO^;
        TESTPARAM[I].ITEM := DIRECTORY.ITEMCODE[I];

        (* Need the parameters only for valid items *)
        IF TESTPARAM[I].ITEM >= 0
          THEN
              BEGIN (2)
                TESTPARAM[I].A := ITEMINFO.A;
                TESTPARAM[I].B := ITEMINFO.B;
                TESTPARAM[I].C := ITEMINFO.C;
              END;  (2)
      END;  (1)
    CLOSE(FILEITEMINFO,LOCK);
  END; (* Param_Array *)

  (* ---------------------------------------------------------------- *)

  (* File containing procedure Verify *)
     (*$I /TFILES/SMGR.DIR/S.VERF.TEXT *)

  (* File containing procedure Newinfo *)
     (*$I /TFILES/SMGR.DIR/S.NEW.TEXT *)

  (* File containing procedure Listinfo *)
     (*$I /TFILES/SMGR.DIR/S.LIST.TEXT *)

  (* File containing procedure Modifyinfo *)
     (*$I /TFILES/SMGR.DIR/S.MODF.TEXT *)

  (* File containing procedure Findinfo *)
     (*$I /TFILES/SMGR.DIR/S.FIND.TEXT *)

  (* File containing procedure Analyze*)
     (*$I /TFILES/SMGR.DIR/S.ANALYZE.TEXT *)

  (* ---------------------------------------------------------------- *)

  (* This procedure shows command level selections *)
  (* It is called from : Procedure Infosetup *)

  PROCEDURE INFOMENU;
  BEGIN
    PAGE(OUTPUT);
    GOTOXY(20,0);
    WRITE('INFOTABLE MANAGER MENU');
    GOTOXY(0,4);
    WRITE('Select one of the following options by entering its number.');
    GOTOXY(16,8);
    WRITE('1.  QUIT');
    GOTOXY(16,9);
    WRITE('2.  MAKE NEW INFO TABLE');
    GOTOXY(16,10);
    WRITE('3.  LIST INFO TABLE');
    GOTOXY(16,11);
    WRITE('4.  VERIFY INFO TABLE');
    GOTOXY(16,12);
    WRITE('5.  MODIFY INFO TABLE');
    GOTOXY(16,13);
    WRITE('6.  FIND ITEMS IN TABLE');
    GOTOXY(16,14);
    WRITE('7.  ANALYZE INFO TABLE ROWS');
    GOTOXY(16,17);
```

```
          WRITE('Enter Choice # : ');
        END; (* Infomenu *)

    (* ................................................................. *)

    BEGIN   (* Infosetup *)
      REPEAT
        ESCPROC := FALSE;
        OTHERINFO := FALSE;
        INFOMENU;
        COMMAND := GETCHAR(['1'..'7','L'],TRUE,FALSE,TRUE);

        CASE COMMAND OF
          '1' : ;
          '2' : NEWINFO;
          '3' : LISTINFO;
          '4' : VERIFY;
          '5' : MODIFYINFO;
          '6' : FINDINFO;
          '7' : ANALYZE;
          'L' : BEGIN {1}
                  OTHERINFO := TRUE;
                  LISTINFO;
                END;  {1}
        END;  (* cases *)
      UNTIL COMMAND = '1';
    END;  (* Infosetup *)

(* ..................................................................... *)

(* main program *)
BEGIN
  DIGITS := ['0'..'9'];
  LETTERS := ['A'..'Z','a'..'z'];
  CHARACTERS := [CHR(32)..CHR(126)];
  FILLCHAR(LINEBUF[0],MAXLINEBUF,' ');
  GETDIRINFO;

  REPEAT
    ESCPROC := FALSE;
    MENU;
    COMMAND := GETCHAR(['1'..'4'],TRUE,FALSE,TRUE);
    CASE COMMAND OF
      '1' : ;
      '2' : INFOSETUP;
      '3' : ;
      '4' : ;
    END;  (* cases *)
  UNTIL COMMAND = '1';

  PAGE(OUTPUT);
  GOTOXY(15,10);
  WRITE('Loading Catproject driver');
  SETCHAIN('CATDATA:CATPROJECT');
END. (* Strategy *)
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : SMGR.DIR/S.UTL.TEXT          Volume : TFILES               *)
(*      Codefile : S.MGR.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(*              DEC. 15, 1982                   NPROC                         *)
(******************************************************************************)

(* This file contains the utility procedures used in S.MGR *)


(* form feeds the printer *)
PROCEDURE TOPOFFORM;
BEGIN
  REWRITE(DEST,UNITNUMPRINTER);
  WRITE(DEST,CHR(12));
  CLOSE(DEST,LOCK);
END;   (* top of form *)

(* -------------------------------------------------------------------------- *)

(**** rings the bell ****)
PROCEDURE SQUAWK;
BEGIN
  WRITE(CHR(BELL));
END;  (* squawk *)

(* -------------------------------------------------------------------------- *)

(**** blank out lines  ****)
PROCEDURE BLANKLINES(START,COUNT,ENDCURSOR : INTEGER);
VAR I : INTEGER;
BEGIN
  GOTOXY(0,START);
  FOR I := 1 TO (COUNT-1) DO
    WRITELN(' ' : 39);
  WRITE(' ':39);
  GOTOXY(0,ENDCURSOR);
END;   (* blanklines *)

(* -------------------------------------------------------------------------- *)

(* read an acceptable character from tne keyboard *)
FUNCTION GETCHAR(OKSET : SETOFCHAR;
                 FLUSHQUEUE,ECHO,BEEP : BOOLEAN) : CHAR;
VAR MASK : PACKED ARRAY[0..0] OF CHAR;
BEGIN
  IF FLUSHQUEUE THEN UNITCLEAR(2); (* flush buffer *)
  REPEAT
    UNITREAD(2,MASK,1);
    IF BEEP AND NOT (MASK[0] IN OKSET) THEN SQUAWK;

  UNTIL MASK[0] IN OKSET;
  IF ECHO AND (MASK[0] IN [CHR(32)..CHR(126)]) THEN
    WRITE(MASK[0]);
  GETCHAR := MASK[0];
END;  (* getchar *)

(* -------------------------------------------------------------------------- *)

(**** display a message/wait for a keystroke ****)
PROCEDURE STALL;
VAR STALLCHAR : CHAR;
BEGIN
  WRITE('Press <RET> to continue ');
  STALLCHAR :=
    GETCHAR([CHR(RET),CHR(ESC)],TRUE,FALSE,TRUE);
  IF STALLCHAR = CHR(ESC) THEN EXIT(PROGRAM);
END;  (* stall *)

(* -------------------------------------------------------------------------- *)
(* read in a string and save in a temporary buffer  *)
PROCEDURE FILLBUF(CHARCNT : INTEGER;
                  OKSET : SETOFCHAR; ERASE : BOOLEAN);
VAR I : INTEGER;
    IOCHAR : CHAR;
BEGIN
  I := 0;
  REPEAT
    IF I > (CHARCNT-1) THEN
      IOCHAR := GETCHAR([CHR(LARROW),CHR(RET)],TRUE,TRUE,TRUE)
    ELSE
    BEGIN (1)
      IOCHAR := GETCHAR(OKSET + [CHR(RET), CHR(LARROW), CHR(RARROW)],
                        TRUE, TRUE, TRUE);
      IF IOCHAR IN OKSET THEN
      BEGIN (2)
        LINEBUF[I] := IOCHAR;
        I := I + 1;
      END;  (2)
    END;  (1)
    IF IOCHAR = CHR(LARROW) THEN
    BEGIN (3)
      IF I = 0 THEN
        SQUAWK
      ELSE
      BEGIN (4)
        WRITE(CHR(LARROW));
        I := I - 1;
        IF ERASE THEN
        BEGIN (5)
          WRITE(' ');
          WRITE(CHR(LARROW));
          LINEBUF[I] := ' ';
        END;  (5)
      END;  (4)
    END   (3)
    ELSE
      IF IOCHAR = CHR(RARROW) THEN
      BEGIN (6)
        WRITE(LINEBUF[I]);
        I := I + 1;
      END;  (6)
  UNTIL IOCHAR = CHR(RET);
END;  (* fillbuf *)

(* ---------------------------------------------------------------------- *)

(**** open a new text file ****)
PROCEDURE  GETNEWFILE;
VAR FILENAME : STRING;
    ERRNUM : INTEGER;

    (* .................................................................. *)

    (**** get a legal filename ****)
    FUNCTION NAMEOK : BOOLEAN;
    VAR I : INTEGER;
    BEGIN
      IF FILENAME = '' THEN
      BEGIN (1)
        FILENAME := DEFAULTFILE;
        GOTOXY(44,0);
        WRITE(FILENAME);
      END   (1)
      ELSE
        IF FILENAME[1] = CHR(esc) THEN EXIT(PROGRAM);
      IF (POS('.TEXT',FILENAME) <> (LENGTH(FILENAME) - 4))
         OR (LENGTH(FILENAME) < 6 ) THEN
           FILENAME := CONCAT(FILENAME,'.TEXT');
    (*$I-*)
      RESET(DEST,FILENAME);
    (*$I+*)
      IF IORESULT = 0 THEN
      BEGIN {2}
        WRITELN;
        WRITELN;
        WRITE('Destroy old ',FILENAME,'?  Press ''N'' or ''Y'' ');
        IF GETCHAR(['y','n','Y','N'],TRUE,TRUE,TRUE) IN ['Y','y'] THEN
        BEGIN {3}
          CLOSE(DEST,PURGE);
          REWRITE(DEST,FILENAME);
          NAMEOK := TRUE;
        END   {3}
        ELSE
          NAMEOK := FALSE;
      END   {2}
      ELSE
      BEGIN {4}
      (*$I-*)
        REWRITE(DEST,FILENAME);
      (*$I+*)
        ERRNUM := IORESULT;
        IF IORESULT <> 0 THEN
        BEGIN {5}
          WRITELN;
          WRITELN;
          WRITELN('Cannot open ',FILENAME,' Io error #',ERRNUM);
          NAMEOK := FALSE;
        END   {5}
        ELSE
          NAMEOK := TRUE;
      END;  {4}
    END;  (* nameok *)

(* ....................................................................... *)

BEGIN  (* getnewfile  *)
  REPEAT
    PAGE(OUTPUT);
    WRITE('Enter output file name, then press <RET>  : ');
    READLN(FILENAME);
  UNTIL NAMEOK;
END;

(* ----------------------------------------------------------------------- *)

(* send control characters to screen *)
PROCEDURE SCRCONTROL(I,J,K : INTEGER);   { PASCAL interface to Screen Control}
VAR N: INTEGER;                          { APPLE III Standard Device Drivers}
   G_ARRAY:PACKED ARRAY[0.. 3] OF 0..255;         {....... Pages 34 to 46.}
BEGIN
  G_ARRAY[0]:= I; G_ARRAY[1]:=J; G_ARRAY[2]:=K;
  UNITWRITE(1,G_ARRAY,3,,12);
END;   (* scrcontrol *)

(* ----------------------------------------------------------------------- *)

(* switch to 40 column screen *)
PROCEDURE TEXT40MODE;
BEGIN
  SCRCONTROL(16,0,28); {Text mode 40BW, followed by clearscreen.}
END;  (* text40mode *)

(* ----------------------------------------------------------------------- *)

(* switch to 80 column screen *)
PROCEDURE TEXT80MODE;
BEGIN
  SCRCONTROL(04,0,0); {Restore Viewport to its original condition.}
  SCRCONTROL(16,2,28); { Text Mode 80, followed by clearscreen.}
END;   (* text80mode *)

(* ----------------------------------------------------------------------- *)

(* turn on reverse video *)
PROCEDURE INVERSE;
```

```
BEGIN
  SCRCONTROL(18,0,0);
END;     (* inverse *)

(* ------------------------------------------------------------------------- *)

(* switch back to normal screen *)
PROCEDURE NORMAL;
BEGIN
  SCRCONTROL(17,0,0);
END;
```

```
(*************************************************************************)
(*                                                                       *)
(*      Textfile : SMGR.DIR/S.NEW.TEXT          Volume : TFILES          *)
(*      Codefile : S.MGR.CODE ('Include' file)  Volume : CATDATA         *)
(*                                                                       *)
(*************************************************************************)
(* File last modified : Feb 23, 1983           NPROC                     *)
(*************************************************************************)
(*                                                                       *)
(* This procedure allows user to make a new infotable for a subtest already *)
(* in the database.  Three options are given:                            *)
(* 1. Calculate the info values for a single ability level. This option will *)
(*    calculate and print out either to the screen or the printer all info *)
(*    values for any selected Theta using the a,b, & c parameters already in *)
(*    the database.                                                      *)
(* 2. Read info table from a textfile.  For this option, the textfile must *)
(*    contain only integers.  The program reads an integer and fills up the *)
(*    infotable by filling in all the columns in a row first, then going on *)
(*    to the next row.  It will tell you if the text file contained the wrong *)
(*    amount of numbers to fill the infotable.                           *)
(* 3. Calculate the infotable automatically.  This option will read the a,b, *)
(*    & c parameters from the database for any given test, and calculate the *)
(*    entire set of info values for each T value.  Then those values are *)
(*    sorted and the items with the largest info values are placed into the *)
(*    row in the infotable for that T.                                   *)
(*                                                                       *)
(*************************************************************************)

(* accepts keyboard input to fill in table *)
PROCEDURE NEWINFO;
VAR EXTRA,
    UNDER,
    X,Y,
    TABVALUE : INTEGER;
    DONE : BOOLEAN;

    (* ................................................................... *)

    (* This procedure opens a new text file *)
    (* This procedure is called from: Procedure Infotextfile *)
    PROCEDURE  GETINPUTFILE;
    VAR FILENAME : STRING;
        ERRNUM : INTEGER;
        NAMEOK : BOOLEAN;
    BEGIN
      NAMEOK := FALSE;
      REPEAT
        WRITE('Enter input filename, then press <RET> : ');
        READLN(FILENAME);
        WRITELN;
        IF FILENAME[1] = CHR(esc) THEN EXIT(PROGRAM);
        IF (POS('.TEXT',FILENAME) <> (LENGTH(FILENAME) - 4))
           OR (LENGTH(FILENAME) < 6 ) THEN
             FILENAME := CONCAT(FILENAME,'.TEXT');
      (*$I-*)
        RESET(DEST,FILENAME);
      (*$I+*)
        IF IORESULT = 0 THEN
          NAMEOK := TRUE
        ELSE
        BEGIN
          ERRNUM := IORESULT;
          WRITELN('IO error #',ERRNUM);
          WRITELN;
        END;
      UNTIL NAMEOK;
    END;  (* getinputfile *)

    (* ------------------------------------------------------------------- *)

    (* This procedure reads an infotable from a textfile and checks to *)
    (* see that it has the correct number of data in it. *)
    (* This procedure is called from: Procedure Newinfo. *)
```

```
PROCEDURE INFOTEXTFILE;
BEGIN
  GETINPUTFILE;
  X := 1;
  Y := 1;
  EXTRA := 0;
  DONE := FALSE;
  PAGE(OUTPUT);
  WRITE('Loading .');
  WHILE NOT EOF(DEST) DO
    BEGIN (*1*)
      WHILE NOT EOLN(DEST) DO
        BEGIN (*2*)
          READ(DEST,TABVALUE);
          IF NOT DONE
            THEN INFOTABLE[X,Y] := TABVALUE
            ELSE EXTRA := EXTRA + 1;
          X := X + 1;
          IF X > INFOCOLUMN
            THEN BEGIN (*3*)
                   WRITE('.');
                   Y := Y + 1;
                   X := 1;
                   IF Y > INFOROW THEN DONE := TRUE;
                 END;  (*3*)
        END;  (*2*)
      READLN(DEST);
    END;  (*1*)
  WRITELN;
  IF (Y = (INFOROW + 1)) AND (X = 1) AND (EXTRA = 0)
    THEN WRITELN('Info table loaded.  No errors.')
    ELSE IF EXTRA > 0
           THEN BEGIN (*4*)
                  WRITE('Info table loaded, ',
                         EXTRA,' extra items in input file.');
                  SQUAWK;
                END   (*4*)
           ELSE BEGIN (*5*)
                  UNDER :=
                    (INFOROW * INFOCOLUMN) - (((Y-1) * INFOCOLUMN) + (X-1));
                  WRITE('Info table loaded, ',
                         UNDER,' items under.  Input not complete');
                  SQUAWK;
                END;  (*5*)
  CLOSE(DEST,LOCK);
  WRITELN;
  WRITELN;
  STALL;
END; (* Infotextfile *)

(* ------------------------------------------------------------------------ *)

(* Bounded exponential function *)
(* This function is called by: Function Info *)
FUNCTION EXPF( X : REAL ) : REAL;
CONST XMAX = 29.0;
      XMIN = -87.0;
VAR Y : REAL;
BEGIN
  IF X < XMIN
    THEN Y := XMIN
    ELSE IF X > XMAX
           THEN Y := XMAX
           ELSE Y := X;
  EXPF := EXP(Y);
END; (* Function Expf *)

(* ------------------------------------------------------------------------ *)

(* Calculate info value from A,B,C parameters for this Theta *)
(* This function is called by: Procedure Optimum *)
FUNCTION INFO( T, A, B, C : REAL) : REAL;
CONST D = 1.7;     (* Scaling Factor *)
```

```
VAR DA, TMP, Y, Z, ZC : REAL;
BEGIN
  DA := D * A;
  Y := DA * (B - T);
  Z := EXPF(Y);
  ZC := Z * C;
  TMP := DA * DA * (Z - ZC) / (1.0 + ZC);
  INFO := TMP * (1 / ((1.0 + Z) * (1.0 + Z)));
END; (* Function Info *)

(* ---------------------------------------------------------------------- *)

(* Quicksort of infovalues puts largest into first array location *)
(* and puts the rest into descending order *)
(* This procedure is called by: Procedure Optimum *)
PROCEDURE SORTINFOVALUES(LEFT,RIGHT : INTEGER);
VAR LPTR,RPTR,           (* Pointers to array locations *)
    TEMPITEM : INTEGER;  (* Temporary during exchange *)
    TEMPINFO,            (*     "       "       "     *)
    PARTITION : REAL;    (* Midpoint of array to which left & right
                            values are compared for sorting  *)
BEGIN
  LPTR := LEFT;
  RPTR := RIGHT;
  PARTITION := SORTARRAY[ (LEFT + RIGHT) DIV 2 ].INFOVALUE;
  (* Divide array into two sections: those infovalues greater than
     the partition value, and those less than the partition value. *)
  REPEAT
    WHILE SORTARRAY[LPTR].INFOVALUE > PARTITION DO
      LPTR := LPTR + 1;
    WHILE PARTITION > SORTARRAY[RPTR].INFOVALUE DO
      RPTR := RPTR - 1;
    IF LPTR <= RPTR
      THEN BEGIN
             (* Exchange, these are on the wrong sides of partition *)
              TEMPITEM := SORTARRAY[LPTR].ITEM;
              TEMPINFO := SORTARRAY[LPTR].INFOVALUE;
              SORTARRAY[LPTR].ITEM := SORTARRAY[RPTR].ITEM;
              SORTARRAY[LPTR].INFOVALUE := SORTARRAY[RPTR].INFOVALUE;
              SORTARRAY[RPTR].ITEM := TEMPITEM;
              SORTARRAY[RPTR].INFOVALUE := TEMPINFO;
              LPTR := LPTR + 1;
              RPTR := RPTR - 1;
            END;
    UNTIL LPTR > RPTR;
    (* Recursively sort each section that was partitioned *)
    IF LEFT < RPTR
      THEN SORTINFOVALUES(LEFT,RPTR);
    IF LPTR < RIGHT
      THEN SORTINFOVALUES(LPTR,RIGHT);
END; (* Sortinfovalues *)

(* ---------------------------------------------------------------------- *)

(* This procedure is for debugging only.  It prints out the SORTARRAY *)
(* so that it can be reviewed for accuracy.  It is for ONE value of T.*)
(* This procedure is called by: Procedure Optimum   *)
PROCEDURE OUTPTINFO(K:INTEGER; T:REAL; TITLE,DESTNAME:STRING);
CONST LINESPERPAGE = 66;
VAR J,LINECOUNT : INTEGER;
    DEST : TEXT;
BEGIN
  REWRITE(DEST,DESTNAME);
  IF TITLE = 'UNSORTED'
    THEN BEGIN
           WRITELN(DEST,
           '       UNSORTED INFOTABLE FOR ',DIRECTORY.TESTNAME,'  T=',T);
           WRITELN(DEST,
           '       ================================================');
         END
    ELSE BEGIN
           WRITELN(DEST,
           '         SORTED INFOTABLE FOR ',DIRECTORY.TESTNAME,'  T=',T);
           WRITELN(DEST,
           '        =================================================');
         END;
  FOR J := 1 TO 4 DO
    WRITE(DEST,'   ITEM INFOVALUE');
  WRITELN(DEST);
  LINECOUNT := 3;
  FOR J := 0 TO K DO
    BEGIN
      IF (J MOD 4) = 0
        THEN BEGIN
               LINECOUNT :=LINECOUNT + 1;
               WRITELN(DEST);
             END;
      WRITE(DEST,SORTARRAY[J].ITEM:7,SORTARRAY[J].INFOVALUE:10:7);
    END;
  IF DESTNAME = 'PRINTER:'
    THEN FOR J := LINECOUNT TO LINESPERPAGE DO
           WRITELN(DEST);
  WRITELN;
  CLOSE(DEST,LOCK);
END; (* Outptinfo *)

(* ----------------------------------------------------------------------- *)

(* Optimize the info -- ie. get valid items and sort by infovalue *)
(* This procedure is called by: Procedure Table *)
PROCEDURE OPTIMUM(T : REAL; ONE : BOOLEAN; DESTNAME : STRING);
VAR I,         (* Loop counter *)
    K,         (* Item counter *)
    MINITEMPOOL : INTEGER;
BEGIN
  MINITEMPOOL := MAXSAMPLES + 1;
  K := 0;
  WRITELN;
  WRITELN('COMPUTING INFOVALUES FOR THETA= ',T);
  FOR I:= MINITEMPOOL TO MAXITEMPOOL DO
    (* Get only valid items, and put into an array with infovalues *)
    IF TESTPARAM[I].ITEM >= 0
      THEN BEGIN
             WRITE('.');
             SORTARRAY[K].ITEM := TESTPARAM[I].ITEM;
             SORTARRAY[K].INFOVALUE := INFO( T, TESTPARAM[I].A,
                                                TESTPARAM[I].B,
                                                TESTPARAM[I].C );
             K := K + 1;
           END;
  WRITELN;
  K := K - 1;
  IF K < INFOCOLUM
    THEN BEGIN
           SQUAWK;
           WRITELN;
           WRITELN('NOT ENOUGH QUESTIONS IN TEST TO MAKE INFOTABLE!!');
           STALL;
           EXIT(NEWINFO);
         END;
  WRITELN;
  WRITELN('SORTING THE INFOTABLE ');
  SORTINFOVALUES(0,K);
  IF ONE
    THEN OUTPTINFO(K,T,'SORTED',DESTNAME);
END; (* Optimum *)

(* ----------------------------------------------------------------------- *)

(* Create infotable *)
(* This procedure is called by: Procedure Newinfo *)
PROCEDURE TABLE;
(* The constants TMIN (Lowest value of ability level) and DT (Increment
   value) are global to program S.MGR *)

VAR I,J : INTEGER;          (* Loop counters *)
```

```
        T : REAL;         (* Theta -- ability level *)
    BEGIN
      T := TMIN;
      FOR J := 1 TO INFOROW DO     (* number of rows in infotable *)
        BEGIN
          OPTIMUM(T,FALSE,'');
          (* after sorted, put into infotable *)
          FOR I := 1 TO INFOCOLUMN DO
            INFOTABLE[I,J] := SORTARRAY[I-1].ITEM;
          T := T + DT;
        END;
    END; (* Table *)

    (* --------------------------------------------------------------------- *)

    (* This allows you to enter from the terminal a single value *)
    (* for T and calculate and print out all infovalues for that T *)
    (* This procedure is called from: Newinfo *)
    PROCEDURE ONETHETA;
    VAR T : REAL;
        MORETHETAS : BOOLEAN;
    BEGIN
      MORETHETAS := TRUE;
      REPEAT
        WRITELN;
        WRITELN('ENTER ABILITY LEVEL (real number between -2.5 and +2.5)');
        WRITE('THETA : ');
        READLN(T);
        WRITELN;WRITELN('WHERE DO YOU WANT THE OUTPUT?');
        WRITELN('    1) SCREEN');
        WRITELN('    2) PRINTER');
        IF GETCHAR(['1','2'],TRUE,FALSE,TRUE) = '1'
          THEN OPTIMUM(T,TRUE,'CONSOLE:')
          ELSE OPTIMUM(T,TRUE,'PRINTER:');
        WRITELN;WRITELN('WANT TO DO ANOTHER ONE?');
        IF GETCHAR(['Y','y','N','n'],TRUE,FALSE,TRUE) IN ['N','n']
          THEN MORETHETAS := FALSE;
      UNTIL NOT MORETHETAS;
    END; (* Onetheta *)

    (* --------------------------------------------------------------------- *)

    (* This procedure updates the info table and then branches to the *)
    (* 'Verify' option to update the second info table.   *)
    PROCEDURE UPDATE;
    BEGIN
      UPDATEINFOFILE(CURRINDEXRECNUM);
      PAGE(OUTPUT);
      CHECKTAB(CURRINDEXRECNUM);
    END;

(* ......................................................................... *)

BEGIN (* new info table *)
  LOADTEST('Make new info table for which test? : ');
  IF ESCPROC THEN
    EXIT(NEWINFO);
  PAGE(OUTPUT);
  GOTOXY(15,0);
  WRITE('MAKE INFOTABLE FOR : ',DIRECTORY.TESTNAME);
  GOTOXY(0,4);
  WRITE('Select one of the following options by entering its number.');
  GOTOXY(12,8);
  WRITE('1. Quit');
  GOTOXY(12,9);
  WRITE('2. Infotable for a single ability level Theta');
  GOTOXY(12,10);
  WRITE('3. Read infotable values from a textfile');
  GOTOXY(12,11);
  WRITE('4. Calculate all infotable values from internal parameters');
  GOTOXY(12,15);
  WRITE('Enter Choice # : ');
  CASE GETCHAR(['1','2','3','4'],TRUE,FALSE,TRUE) OF
```

```
    '1': ;
    '2': BEGIN
           PAGE(OUTPUT);
           PARAM_ARRAY;
           PAGE(OUTPUT);
           ONETHETA;
         END;
    '3': BEGIN
           PAGE(OUTPUT);
           INFOTEXTFILE;
           UPDATE;
         END;
    '4': BEGIN
           PAGE(OUTPUT);
           PARAM_ARRAY;
           PAGE(OUTPUT);
           TABLE;
           UPDATE;
         END;
  END; (* Cases *)
END;  (* new info *)
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : SMGR.DIR/S.VERF.TEXT         Volume : TFILES               *)
(*      Codefile : S.MGR.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(* File last modified : JULY 7, 1983            NPROC                         *)
(******************************************************************************)

(* checks the infotable for duplications or missing entries *)
PROCEDURE CHECKTAB(INFOREC : INTEGER);
VAR ERRORCNT,
  DSLOT,
  I,J,K,COUNT,OFFSET: INTEGER;
  LISTERR : BOOLEAN;
  DUPLIC   : ARRAY[1..720] OF RECORD
                               RW,
                               IINDEX,
                               KINDEX,
                               ITEM:INTEGER;
                             END;
  ERRLIST  : ARRAY[1..720] OF RECORD
                               R,
                               C,
                               CODE : INTEGER;
                             END;
BEGIN
  PAGE(OUTPUT);
  LOADINFO(INFOREC);
  WRITELN('Verifying ');
  ERRORCNT := 0;
  COUNT:=0;

  FOR J := 1 TO INFOROW+1 DO
  BEGIN
    FOR I := 1 TO INFOCOLUMN DO
    BEGIN
      IF J < INFOROW +1 THEN
      BEGIN
        WRITE('.');
        IF INFOTABLE[I,J] < 0 THEN
          DSLOT := -1
        ELSE
          DSLOT := SLOT(INFOTABLE[I,J]); (* Transforms question # to pointer *)
        IF DSLOT < 0 THEN
        BEGIN
          ERRORCNT := ERRORCNT + 1;
          WITH ERRLIST[ERRORCNT] DO
          BEGIN
             R := J;
             C := I;
             CODE := INFOTABLE[I,J];
          END;
        END;
        INFOTABLE[I,J] := DSLOT;
      END;

      (* Checks for duplicate entries in the infotable that are not equal*)
      (* to -1.  The entry and the row location are saved and displayed  *)
      (* later in the routine.  The second (pointer)infotable is checked *)
      FOR K:=1 TO 20 DO
      BEGIN
        IF J > 1 THEN
        BEGIN
          UFFSET:=J-1;
          IF INFOTABLE[I,OFFSET]=INFOTABLE[K,OFFSET] THEN
          BEGIN
            IF (I > K) AND (INFOTABLE[K,OFFSET] > 0) THEN
            BEGIN
              COUNT:=COUNT+1;
              WITH DUPLIC[COUNT] DO
              BEGIN
                RW:=OFFSET;
                ITEM:=INFOTABLE[K,OFFSET];
                IINDEX:=I;
                KINDEX:=K;
              END;
            END
          END;
        END;
      END;
    END;
    IF J MOD 3 = 0 THEN WRITELN;
  END;
  WRITELN;
  WRITELN;
  WRITELN('There are ',ERRORCNT,' errors in the info tables.');
  WRITELN;
  IF ERRORCNT > 0 THEN
  BEGIN
    WRITE('List the errors?  Press ''Y'' or ''N'' : ');
    IF GETCHAR(['y','n','Y','N'] ,true,false,true) IN ['Y','y'] THEN
    BEGIN
      PAGE(OUTPUT);
      FOR I := 1 TO ERRORCNT DO
      BEGIN
         WITH ERRLIST[I] DO
         BEGIN
           WRITELN('Error! Infotable entry ',CODE,
                 ' at row ',R,' ,column ',C,' not found in directory.');
         END;
         IF I MOD 20 = 0 THEN
         BEGIN
           WRITELN;
           STALL;
           WRITELN;
           WRITELN;
         END;
      END;
    END
    ELSE
      WRITELN;
  END;
  IF COUNT > 0 THEN
  BEGIN
    WRITELN('DUPLICATION ERRORS EXIST IN THE SUBTEST INFOTABLE');
    WRITELN;
    FOR K:=1 TO COUNT DO
    BEGIN
      WITH DUPLIC[K] DO
        WRITELN('ITEM ',ITEM,' DUPLICATED IN ROW ',RW,' COLUMNS ',IINDEX,' ',KINDEX);
    END;
  END;
  STALL;
  UPDATEINFO(INFOREC + MAXSUBTESTS + 1);
END;  (* checktab *)


(* This procedure stores the index of the directory of the question rather    *)
(* than the question code.  This will cut down search time for item in direc- *)
(* tory when referencing infotable.  It is also a verification that all items *)
(* in the info table are also in the subtest directories.                     *)
(* This procedure is called from : Procedure Infosetup  *)
PROCEDURE VERIFY;
BEGIN
  LOADTEST('Verify infotable for which subtest? : ');
  IF ESCPROC THEN
    EXIT(VERIFY);
  CHECKTAB(CURRINDEXRECNUM);
END;  (* Verify   *)
```

```
(********************************************************************************)
(*                                                                              *)
(*      Textfile : SMGR.DIR/S.MODF.TEXT          Volume : TFILES                *)
(*      Codefile : S.MGR.TEXT ('Include' file)  Volume : CATDATA                *)
(*                                                                              *)
(********************************************************************************)
(* File last modified : Feb 23, 1983             NPROC                          *)
(********************************************************************************)

(* allows modifications to information table *)
PROCEDURE MODIFYINFO;
VAR X,Y,
    TABVALUE : INTEGER;
    MODCHAR : CHAR;


(* ........................................................................... *)

BEGIN (* modify info *)
  LOADTEST('Modify infotable for which test? : ');
  IF ESCPROC THEN
    EXIT(MODIFYINFO);
  LOADINFO(CURRINDEXRECNUM);
  FORMATSCREEN;
  GOTOXY(0,0);
  WRITE('Test : ',DIRECTORY.TESTNAME);
  Y := 1;
  X := 1;
  GOTOXY(50,2);
  WRITE('INSTRUCTIONS');
  GOTOXY(47,4);
  WRITE('Use  --> and <-- to move');
  GOTOXY(47,5);
  WRITE('horizontally in infotable');
  GOTOXY(47,7);
  WRITE('Use <up arrow> and ');
  GOTOXY(47,8);
  WRITE('<down arrow> to move');
  GOTOXY(47,9);
  WRITE('vertically in infotable.');
  GOTOXY(47,11);
  WRITE('To change a value in the');
  GOTOXY(47,12);
  WRITE('table, go to the location');
  GOTOXY(47,13);
  WRITE('you wish to change, Press "M",');
  GOTOXY(47,14);
  WRITE('enter the new value, then');
  GOTOXY(47,15);
  WRITE('press <RET>.');
  GOTOXY(47,17);
  WRITE('Press <ESC> to quit without');
  GOTOXY(47,18);
  WRITE('updating.');
  GOTOXY(47,20);
  WRITE('Press <CNTRL-C> to quit and');
  GOTOXY(47,21);
  WRITE('update');
  REPEAT
    GOTOXY(6,5);
    WRITE(Y,'  ');
    GOTOXY(9,7);
    WRITE(X,'  ');
    GOTOXY(10,9);
    WRITE('      ');
    GOTOXY(10,9);
    WRITE(INFOTABLE[X,Y]);
    GOTOXY(X+17,(((Y-1) DIV 2) + 3));
    MODCHAR := GETCHAR([CHR(LARROW),CHR(RARROW),CHR(UP),CHR(DOWN),
                        CHR(ESC),CHR(ETX),'M','m'],TRUE,FALSE,TRUE);
    CASE ORD(MODCHAR) OF
      LARROW : IF X = 1 THEN
                 BEGIN
                 IF Y > 1 THEN
                 BEGIN
                   Y := Y - 1;
                   X := INFOCOLUMN;
                 END
                 ELSE
                   SQUAWK;
               END
               ELSE
                 X := X - 1;

      RARROW : IF X = INFOCOLUMN THEN
               BEGIN
                 IF Y < INFOROW THEN
                 BEGIN
                   Y := Y + 1;
                   X := 1;
                 END
                 ELSE
                   SQUAWK;
               END
               ELSE
                 X := X + 1;

      UP     : IF Y = 1 THEN
                 SQUAWK
               ELSE
                 Y := Y - 1;

      DOWN   : IF Y = INFOROW THEN
                 SQUAWK
               ELSE
                 Y := Y + 1;
      ESC    : EXIT(MODIFYINFO);

      ETX    : ;

    END;  (* case *)
    IF MODCHAR IN ['M','m'] THEN
    BEGIN
      GOTOXY(10,9);
      WRITE('        ');
      GOTOXY(10,9);
      READLN(TABVALUE);
      INFOTABLE[X,Y] := TABVALUE;
    END;
  UNTIL ORD(MODCHAR) = ETX;
  UPDATEINFOFILE(CURRINDEXRECNUM);
  CHECKTAB(CURRINDEXRECNUM);
END;  (* modify info *)
```

```
(*************************************************************************)
(*                                                                       *)
(*      Textfile : SMGR.DIR/S.LIST.TEXT          Volume : TFILES         *)
(*      Codefile : S.MGR.CODE ('Include' file)   Volume : CATDATA        *)
(*                                                                       *)
(*************************************************************************)
(*                DEC. 14, 1982                  NPRDC                   *)
(*************************************************************************)

(* lists the information table to file or printer *)
PROCEDURE LISTINFO;
VAR I,J : INTEGER;
    PRINTER,
    TOFILE : BOOLEAN;
    SELECT : CHAR;

    (* ................................................................ *)

    PROCEDURE LISTPRINTER;
    BEGIN
      PAGE(OUTPUT);
      WRITE('Writing .');
      WRITELN(DEST,'Test : ',DIRECTORY.TESTNAME);
      WRITELN(DEST);
      J := 1;
      WRITE(DEST,'    ');
      FOR I := 1 TO 10 DO WRITE(DEST,I : 7);
      WRITELN(DEST);
      WRITELN(DEST,
'    __________________________________________________________________________');
      REPEAT
        WRITE('.');
        WRITE(DEST,J:2,'| ');
        FOR I := 1 TO 10 DO
          WRITE(DEST,INFOTABLE[I,J] : 7);
        WRITELN(DEST);
        J := J + 1;
      UNTIL J > INFOROW;
      WRITELN(DEST);
      WRITELN(DEST);
      WRITELN(DEST);
      WRITELN(DEST);
      J := 1;
      WRITE(DEST,'    ');
      FOR I := 11 TO INFOCOLUMN DO WRITE(DEST,I : 7);
      WRITELN(DEST);
      WRITELN(DEST,
'    __________________________________________________________________________');
      REPEAT
        WRITE('.');
        WRITE(DEST,J:2,'| ');
        FOR I := 11 TO INFOCOLUMN DO
          WRITE(DEST,INFOTABLE[I,J] : 7);
        WRITELN(DEST);
        J := J + 1;
      UNTIL J > INFOROW;
      WRITELN(DEST);
      CLOSE(DEST,LOCK);
    END;  (* listprinter *)

    (* ---------------------------------------------------------------- *)

    (* list to console *)
    PROCEDURE LISTCONSOLE;

    (* ................................................................ *)

        PROCEDURE L1;
          BEGIN
            PAGE(OUTPUT);
            WRITELN('Test : ',DIRECTORY.TESTNAME);
            WRITELN;
            WRITE('    ');
            FOR I := 1 TO 10 DO WRITE(I : 7);
            WRITELN;
            WRITELN(
'_________________________________________________________________________');
            FOR J := 1 TO 18 DO
            BEGIN
              WRITE(J:2,'| ');
              FOR I := 1 TO 10 DO
                WRITE(INFOTABLE[I,J] : 7);
              WRITELN;
            END;
            GOTOXY(0,23);
            STALL;
          END;  (* L1 *)

          (* ---------------------------------------------------------------- *)

          PROCEDURE L2;
          BEGIN
            PAGE(OUTPUT);
            WRITELN('Test : ',DIRECTORY.TESTNAME);
            WRITELN;
            WRITE('    ');
            FOR I := 1 TO 10 DO WRITE(I : 7);
            WRITELN;
            WRITELN(
'_________________________________________________________________________');
            FOR J := 19 TO 36 DO
            BEGIN
              WRITE(J:2,'| ');
              FOR I := 1 TO 10 DO
                WRITE(INFOTABLE[I,J] : 7);
              WRITELN;
            END;
            GOTOXY(0,23);
            STALL;
          END;   (* L2 *)

    (* ................................................................... *)

    BEGIN  (* listconsole *)
      L1;
      L2;
      PAGE(OUTPUT);
      WRITELN('Test : ',DIRECTORY.TESTNAME);
      WRITELN;
      WRITE('    ');
      FOR I := 11 TO INFOCOLUMN DO WRITE(I : 7);
      WRITELN;
      WRITELN(
'_________________________________________________________________________');
      FOR J := 1 TO 18 DO
      BEGIN
        WRITE(J:2,'| ');
        FOR I := 11 TO INFOCOLUMN DO
          WRITE(INFOTABLE[I,J] : 7);
        WRITELN;
      END;
      GOTOXY(0,23);
      STALL;
      PAGE(OUTPUT);
      WRITELN('Test : ',DIRECTORY.TESTNAME);
      WRITELN;
      WRITE('    ');
      FOR I := 11 TO INFOCOLUMN DO WRITE(I : 7);
      WRITELN;
      WRITELN(
'_________________________________________________________________________');
      FOR J := 19 TO 36 DO
      BEGIN
        WRITE(J:2,'| ');
        FOR I := 11 TO INFOCOLUMN DO
          WRITE(INFOTABLE[I,J] : 7);
```

```
   WRITELN;
   END;
   GOTOXY(0,23);
   STALL;
  END;  (* listconsole *)

(* ........................................................................ *)

 (* writes info tables to individual files for floppy version of test *)
 (* administration.  Keeps format same.                                *)
 PROCEDURE WRITETOFILE;
 VAR FILENAME : STRING;
   SMALLINFO : FILE OF TABLE;
 BEGIN
  PAGE(OUTPUT);
  WRITE('Enter two character code for subtest, then press <RET>  : ');
  READLN(FILENAME);
  IF FILENAME = '' THEN
   EXIT(WRITETOFILE)
  ELSE
   IF FILENAME[1] = CHR(esc) THEN
    EXIT(WRITETOFILE);
  FILENAME := CONCAT(FILENAME,'.INFO.DATA');
  REWRITE(SMALLINFO,FILENAME);
  LOADINFO(CURRINDEXRECNUM);
  SMALLINFO^ := INFOTABLE;
  SEEK(SMALLINFO,0);
  PUT(SMALLINFO);
  LOADINFO(CURRINDEXRECNUM + MAXSUBTESTS + 1);
  SMALLINFO^ := INFOTABLE;
  SEEK(SMALLINFO,1);
  PUT(SMALLINFO);
  CLOSE(SMALLINFO,LOCK);
 END;  (* write to file *)



BEGIN (* Listinfo *)
 PAGE(OUTPUT);
 GOTOXY(20,0);
 WRITE('SELECT OUTPUT MENU');
 GOTOXY(0,4);
 WRITE('Select one of the following options by entering its number.');
 GOTOXY(16,8);
 WRITE('1.  QUIT');
 GOTOXY(16,9);
 WRITE('2.  LIST INFOTABLE TO SCREEN');
 GOTOXY(16,10);
 WRITE('3.  LIST INFOTABLE TO PRINTER');
 GOTOXY(16,11);
 WRITE('4.  LIST INFOTABLE TO FLOPPY DATA FILE');
 GOTOXY(16,12);
 WRITE('5.  LIST INFOTABLE TO TEXT FILE');
 GOTOXY(16,14);
 WRITE('Enter choice # : ');
 PRINTER := FALSE;
 TOFILE := FALSE;
 SELECT := GETCHAR(['1'..'5'],TRUE,FALSE,TRUE);
 CASE SELECT OF
  '1' : EXIT(LISTINFO);
  '2' : ;
  '3' : BEGIN
      PRINTER := TRUE;
      REWRITE(DEST,'PRINTER:');
    END;
  '4' : TOFILE := TRUE;

  '5' : BEGIN
      GETNEWFILE;
      PRINTER := TRUE;
```

```
          END;
  END;
  LOADTEST('List infotable for which subtest? : ');
  IF ESCPROC THEN
    EXIT(LISTINFO);

  IF TOFILE THEN
  BEGIN
    WRITETOFILE;
    EXIT(LISTINFO);
  END;

  IF OTHERINFO THEN
    LOADINFO(CURRINDEXRECNUM + MAXSUBTESTS + 1)
  ELSE
    LOADINFO(CURRINDEXRECNUM);
  IF PRINTER THEN
    LISTPRINTER
  ELSE
    LISTCONSOLE;
END;  (* list info *)
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : SMGR.DIR/S.FIND.TEXT         Volume : TFILES               *)
(*      Codefile : S.MGR.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(*              DEC. 14, 1982                   NPRDC                         *)
(******************************************************************************)

(* This procedure allows you to examine automatically the infotable for *)
(* a specific item to see if it is ever called.  Or, it will print out  *)
(* all of the items that are never called. *)
(* This procedure is called by: Procedure *)
PROCEDURE FINDINFO;

    (* ...................................................................... *)

    PROCEDURE FINDMENU;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(20,0);
      WRITE('menu');
      GOTOXY(0,4);
      WRITE('Select one of the following options by entering its number:');
      GOTOXY(14,8);
      WRITE('1.  QUIT');
      GOTOXY(14,9);
      WRITE('2.  Find a single item in infotable');
      GOTOXY(14,10);
      WRITE('3.  List to screen all occurrences of an item');
      GOTOXY(14,11);
      WRITE('4.  List to printer all occurrences of an item');
      GOTOXY(14,12);
      WRITE('5.  List to screen all items not used');
      GOTOXY(14,13);
      WRITE('6.  List to printer all items not used');
      GOTOXY(14,17);
      WRITE('Enter Choice # : ');
    END; (* Findmenu *)

    (* ---------------------------------------------------------------------- *)

    (* This procedure will find a single item if it is in the infotable. *)
    (* This procedure is called from: Procedure Findinfo *)
    PROCEDURE FINDITEM(DESTNAME:STRING);
    VAR N,X,Y,ITEMNUM : INTEGER;
        MOREITEMS,FOUND : BOOLEAN;
        DEST : TEXT;

        (* .................................................................. *)

        PROCEDURE ITEMFOUND(VAR N,X,Y : INTEGER; DESTNAME : STRING);
        BEGIN
          IF DESTNAME = 'NOWHERE'
            THEN BEGIN (*1*)
                   WRITELN;
                   WRITELN('ITEM : ',ITEMNUM,'    ',
                           'ROW : ',X,'    ','COLUMN : ',Y);
                   WRITELN;WRITELN('Do you want the next location?');
                   IF GETCHAR(['Y','N','y','n'],TRUE,FALSE,TRUE)
                     IN ['N','n'] THEN BEGIN (*2*)
                                         X := INFOROW;
                                         Y := INFOCOLUMN;
                                       END;  (*2*)
                 END   (*1*)
            ELSE BEGIN (*3*)
                   IF (N MOD 6) = 0
                     THEN WRITELN(DEST);
                   N := N+1;
                   WRITE(DEST,X:3,Y:4,'     ');
                 END;  (*3*)
        END;  (* Itemfound *)
```

```
  (* ................................................................... *)

BEGIN  (* Finditem *)
  REPEAT
    PAGE(OUTPUT);
    WRITE('Item number you wish to find : ');
    READLN(ITEMNUM);
    WRITELN;
    IF NOT (DESTNAME = 'NOWHERE')
      THEN BEGIN
             REWRITE(DEST,DESTNAME);
             WRITELN(DEST);WRITELN(DEST);WRITELN(DEST);
             WRITELN(DEST,'SUBTEST : ',DIRECTORY.TESTNAME);
             WRITELN(DEST,'Locations in infotable of item : ',ITEMNUM);
             WRITELN(DEST);
             FOR N := 1 TO 6 DO
                WRITE(DEST,'ROW','  COL      ');
             N := 0;
           END;
    Y:=1;
    WHILE Y <= INFOCOLUMN DO
      BEGIN  (*1*)
        X := 1;
        WHILE X <= INFOROW DO
          BEGIN (*2*)
            IF ITEMNUM = INFOTABLE[Y,X]
              THEN BEGIN  (*3*)
                     FOUND := TRUE;
                     ITEMFOUND(N,X,Y,DESTNAME);
                   END    (*3*)
              ELSE FOUND := FALSE;
            X := X+1;
          END;  (*2*)
        Y := Y+1;
      END;    (*1*)
    IF NOT (DESTNAME = 'NOWHERE')
      THEN BEGIN
             WRITELN(DEST);
             CLOSE(DEST,NORMAL);
           END;
    IF NOT FOUND THEN WRITELN('No more locations of item ',ITEMNUM);
    WRITELN;WRITELN('Do you want to find a different item?');
    IF GETCHAR(['Y','y','N','n'],TRUE,FALSE,TRUE) IN ['Y','y']
      THEN MOREITEMS := TRUE
      ELSE MOREITEMS := FALSE;
  UNTIL NOT MOREITEMS;
END; (* Finditem *)

 (* ------------------------------------------------------------------- *)

PROCEDURE FINDUNUSED(DESTNAME : STRING);
(* The constants MAXITEMPOOL and MAXSAMPLES are global to Program
   Strategy. *)
TYPE ARRAYTYPE = PACKED ARRAY[0..MAXITEMPOOL] OF INTEGER;
VAR I,J,K,NUMITEMS,MINITEMPOOL : INTEGER;
    NOTFOUND : ARRAYTYPE;
    SORTARRAY : ARRAYTYPE;
    FOUNDARRAY : ARRAYTYPE;
    DEST : TEXT;

    (* ............................................................. *)

    (* This procedure does a quicksort of the SORTARRAY which contains *)
    (* the valid item numbers for this subtest.  The array was filled  *)
    (* in Procedure GETITEMS.  It is a recursive procedure.            *)
    (* This procedure is called by: Procedure Getitems.                *)
    PROCEDURE SORTITEMS(LEFT,RIGHT : INTEGER);
    VAR LPTR,RPTR,              (* Pointers to array locations *)
        TEMP : INTEGER;         (* Temporary during exchange *)
        PARTITION : REAL;       (* Midpoint of array to which left & right
                                   values are compared for sorting *)
    BEGIN
  LPTR := LEFT;
  RPTR := RIGHT;
  PARTITION := SORTARRAY[ (LEFT + RIGHT) DIV 2 ];
  (* Divide array into two sections: those item numbers greater than
     the partition value, and those less than the partition value. *)
  REPEAT
    WHILE SORTARRAY[LPTR] < PARTITION DO
      LPTR := LPTR + 1;
    WHILE PARTITION < SORTARRAY[RPTR] DO
      RPTR := RPTR - 1;
    IF LPTR <= RPTR
      THEN BEGIN
             (* Exchange, these are on the wrong sides of partition *)
             TEMP := SORTARRAY[LPTR];
             SORTARRAY[LPTR] := SORTARRAY[RPTR];
             SORTARRAY[RPTR] := TEMP;
             LPTR := LPTR + 1;
             RPTR := RPTR - 1;
           END;
  UNTIL LPTR > RPTR;
  (* Recursively sort each section that was partitioned *)
  IF LEFT < RPTR
    THEN SORTITEMS(LEFT,RPTR);
  IF LPTR < RIGHT
    THEN SORTITEMS(LPTR,RIGHT);
END; (* Sortitems *)

(* ------------------------------------------------------------------ *)

(* This procedure gets the item numbers from the file.  They are in  *)
(* the file's window variable DIRECTORY (global to Program STRATEGY) *)
(* which was initialized in Procedure LOADTEST.  Since items in that *)
(* file of value -1 indicate a blank record, only the items greater  *)
(* than -1 are loaded into an array to be sorted.                    *)
(* This procedure is called by: Procedure FINDUNUSED.                *)
PROCEDURE GETITEMS(VAR NUMITEMS : INTEGER);
VAR I,MINITEMPOOL : INTEGER;
BEGIN
  PAGE(OUTPUT);         (* Utility procedure in file: T-UTL.TEXT *)
  WRITELN('Getting the items from file');
  MINITEMPOOL := MAXSAMPLES + 1;
  NUMITEMS:= 0;
  FOR I := MINITEMPOOL TO MAXITEMPOOL DO
    IF DIRECTORY.ITEMCODE[I] >= 0
      THEN BEGIN (*1*)
             SORTARRAY[NUMITEMS] := DIRECTORY.ITEMCODE[I];
             NUMITEMS := NUMITEMS + 1;
           END;  (*1*)
  WRITELN('Number of items in file : ',NUMITEMS);
  WRITELN;
  IF NUMITEMS > 0
    THEN NUMITEMS := NUMITEMS - 1
    ELSE BEGIN
           WRITELN(DEST);
           WRITELN(DEST,'There are no items in ',DIRECTORY.TESTNAME);
           EXIT(FINDUNUSED);
         END;
  IF NUMITEMS > 0 THEN begin
                       WRITELN('Sorting items...');
                       SORTITEMS(0,NUMITEMS);
                     end;
END; (* Getitems *)

(* ------------------------------------------------------------------ *)

FUNCTION INTABLE(ITEM : INTEGER) : BOOLEAN;
(* The constants INFOROW and INFOCOLUMN are defined in Procedure *)
(* INFOSETUP.  *)
VAR ROW,COL : INTEGER;
    FOUND : BOOLEAN;
BEGIN
  ROW := 1;
  FOUND := FALSE;
```

```
          REPEAT
            COL := 1;
            REPEAT
              IF ITEM = INFOTABLE[COL,ROW]
                THEN FOUND := TRUE
                ELSE COL := COL + 1;
            UNTIL FOUND OR (COL > INFOCOLUMN);
            ROW := ROW + 1;
          UNTIL FOUND OR (ROW > INFOROW);
          INTABLE := FOUND;
        END; (* Intable *)

    (* ................................................................. *)

    BEGIN  (* Findunused *)
      PAGE(OUTPUT);
      REWRITE(DEST,DESTNAME);
      GETITEMS(NUMITEMS);         (* Count the items as you get & sort them *)
      J := 0;  K := 0;
      LOADINFO(CURRINDEXRECNUM);      (* Loads the infotable from the file into *)
           (* the array variable INFOTABLE as defined in Procedure INFOSETUP *)
      WRITELN;
      WRITELN('Be patient, I have to compare each of these ',NUMITEMS + 1,
                                                      ' items to the infotable.');
      FOR I := 0 TO NUMITEMS DO
        IF INTABLE(SORTARRAY[I])
          THEN BEGIN       (* Put items found in infotable into one array *)
                 WRITE('.');
                 FOUNDARRAY[J] := SORTARRAY[I];
                 J := J + 1;
               END
          ELSE BEGIN            (* Put items not found into another array *)
                 WRITE('.');
                 NOTFOUND[K] := SORTARRAY[I];
                 K := K + 1;
               END;
      WRITELN(DEST);WRITELN(DEST);
      IF J > 0
        THEN BEGIN
               J := J - 1;
               WRITELN(DEST,'List of items found in infotable for : ',
                                                           DIRECTORY.TESTNAME);
               FOR I := 0 TO J DO
                 BEGIN
                   IF (I MOD 10) = 3 THEN WRITELN(DEST);
                   WRITE(DEST,FOUNDARRAY[I]:7);
                 END;
             END
        ELSE WRITELN(DEST,'There are no items in the infotable for : ',
                                                           DIRECTORY.TESTNAME);
      WRITELN(DEST);WRITELN(DEST);
      IF K > 0
        THEN BEGIN
               K := K - 1;
               WRITELN(DEST,'List of items not used in : ',
                                                           DIRECTORY.TESTNAME);
               FOR I := 0 TO K DO
                 BEGIN
                   IF (I MOD 10) = 0 THEN WRITELN(DEST);
                   WRITE(DEST,NOTFOUND[I]:7);
                 END;
             END
        ELSE WRITELN(DEST,'All items are used in : ',DIRECTORY.TESTNAME);
      WRITELN(DEST);WRITELN(DEST);
      CLOSE(DEST,NORMAL);
      IF (DESTNAME = 'CONSOLE:')  THEN STALL;
    END; (* Findunused *)

(* ..................................................................... *)

BEGIN  (* Findinfo *)
  LOADTEST('Find items for which subtest? :');
  IF ESCPROC THEN EXIT(FINDINFO);
  PAGE(OUTPUT);
  LOADINFO(CURRINDEXRECNUM);
  FINDMENU;
  CASE GETCHAR(['1'..'6'],TRUE,FALSE,TRUE) OF
     '1' : ;
     '2' : FINDITEM('NOWHERE');
     '3' : FINDITEM('CONSOLE:');
     '4' : FINDITEM('PRINTER:');
     '5' : FINDUNUSED('CONSOLE:');
     '6' : FINDUNUSED('PRINTER:');
  END; (* cases *)
END;  (* Findinfo *)
```

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : SMGR.DIR/S.ANALYZE.TEXT      Volume : TFILES               *)
(*      Codefile : S.MGR.CODE ('Include' file)  Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(*              DEC. 21, 1982                 NPROC                           *)
(******************************************************************************)
(*                                                                            *)
(*  This procedure allows a user to examine each row of the info table.  A    *)
(*  subtest is selected, and then the destination of the output is entered;   *)
(*  either the console, a printer, or a file.  The row being analyzed is      *)
(*  displayed, followed by a table with the following entries: the column     *)
(*  number, the item code number from the infotable, and the infovalue as     *)
(*  calculated from the A, B, and C parameters.  The formula for this calcu-  *)
(*  lation is in procedure INFO.  After displaying the table for twenty       *)
(*  columns, a sum is displayed.  This is the sum of all of the infovalues    *)
(*  for the particular row being analyzed.  Then the next row is displayed    *)
(*  etc. until all of the rows of the infotable for this subtest have been    *)
(*  shown.  An option is available to display only the sums for each row,     *)
(*  eliminating all of the individual item codes and infovalues.              *)
(*                                                                            *)
(******************************************************************************)


(* This is an 'include' file for program S.MGR. *)
PROCEDURE ANALYZE;
VAR ROW,
    COLUMN,
    HEADR,
    DIRSLOT,
    DATAREC,
    ITEMS,
    ICODE : INTEGER;
    A, B, C,
    T,
    INFOVAL : REAL;
    SUM : PACKED ARRAY[1..INFOROW] OF REAL;
    SELECT : CHAR;
    PRINTALL : BOOLEAN;

    (* ................................................................... *)

    (* Bounded exponential function *)
    (* This function is called by : Function Info *)
    FUNCTION EXPF( X : REAL ) : REAL;
    CONST XMAX = 29.0;
          XMIN = -87.0;
    VAR Y : REAL;
    BEGIN
      IF X < XMIN THEN
        Y := XMIN
      ELSE IF X > XMAX THEN
        Y := XMAX
      ELSE
        Y := X;
      EXPF := EXP(Y);
    END;  (* Function Expf *)

    (* ------------------------------------------------------------------- *)

    (* Calculate info value from A, B, C parameters for this Theta. *)
    (* This function is called by : Procedure ANALYZE. *)
    FUNCTION INFO( T, A, B, C : REAL ) : REAL;
    CONST D = 1.7;  (* Scaling Factor *)
    VAR DA,
        TMP,
        Y,
        Z,
        ZC : REAL;
    BEGIN
      DA := D * A;
      Y := DA * (B - T);
    Z := EXPF(Y);
    ZC := Z * C;
    TMP := DA * DA * (Z - ZC) / (1.0 + ZC);
    INFO := TMP * (1 / ((1.0 + Z) * (1.0 + Z)));
  END;  (* Function Info *)

  (* ---------------------------------------------------------------------- *)

  (* get the output destination *)
  (* this procedure is called by : Procedure ANALYZE *)
  PROCEDURE GETOUTPUTDEST;
  BEGIN
    PAGE(OUTPUT);
    GOTOXY(20,0);
    WRITE('SELECT OUTPUT MENU');
    GOTOXY(0,4);
    WRITE('Select one of the following options by entering its number.');
    GOTOXY(16,8);
    WRITE('1.  QUIT');
    GOTOXY(16,9);
    WRITE('2.  LIST ROW DATA TO SCREEN');
    GOTOXY(16,10);
    WRITE('3.  LIST ROW DATA TO PRINTER');
    GOTOXY(16,11);
    WRITE('4.  LIST ROW DATA TO TEXT FILE');
    GOTOXY(16,14);
    WRITE('Enter choice # : ');

    SELECT := GETCHAR(['1'..'4'],TRUE,FALSE,TRUE);
    CASE SELECT OF
      '1' : EXIT(ANALYZE);
      '2' : REWRITE(DEST,'CONSOLE:');
      '3' : REWRITE(DEST,'PRINTER:');
      '4' : GETNEWFILE;  (* Procedure located in textfile S.UTL *)
    END; (* cases *)
  END;  (* getoutputdest *)

  (* ---------------------------------------------------------------------- *)

  (* Get level of print detail, ie, either all information, or just the    *)
  (* sums of the rows.  *)
  (* This function is called by : Procedure Analyze. *)

  FUNCTION PRINTDETAIL : BOOLEAN;
  VAR OPTION : CHAR;
  BEGIN
    PAGE(OUTPUT);
    GOTOXY(20,0);
    WRITE('SELECT OUTPUT MENU');
    GOTOXY(0,4);
    WRITE('Select one of the following options by entering its number.');
    GOTOXY(16,8);
    WRITE('1.  QUIT');
    GOTOXY(16,9);
    WRITE('2.  LIST ALL ROW DATA');
    GOTOXY(16,10);
    WRITE('3.  LIST ONLY THE SUM OF EACH ROW');
    GOTOXY(16,13);
    WRITE('Select choice # :');

    OPTION := GETCHAR(['1'..'3'],TRUE,FALSE,TRUE);
    CASE OPTION OF
      '1' : EXIT(ANALYZE);
      '2' : PRINTDETAIL := TRUE;
      '3' : PRINTDETAIL := FALSE;
    END; (* cases *)
  END;  (* Printdetail *)

  (* ---------------------------------------------------------------------- *)

  (* This procedure prints error messages if there is incorrect infotable *)
  (* data, and then it exits from procedure ANALYZE. *)
  (* This procedure is called by : Procedure ANALYZE. *)
```

```
PROCEDURE ERROR(ERTYPE, NUMBER : INTEGER);
BEGIN
  (* SQUAWK is located in S.UTLS -- it rings the bell. *)
  SQUAWK;

  CASE ERTYPE OF
     1 : BEGIN  (* DIRSLOT will cause a value range error *)
           IF NUMBER < 0 THEN
             WRITELN('NO INFOTABLE EXISTS FOR ',DIRECTORY.TESTNAME);
           IF NUMBER > 300 THEN
             WRITELN('INFOTABLE POINTER IS TOO BIG!!    VERIFY IT.');
         END;

     2 : BEGIN (* ICODE will cause a value range error *)
           IF NUMBER < 0 THEN
             WRITELN('INCORRECT ITEMCODE!  IT WAS ',NUMBER);
         END;
  END; (* cases *)
  WRITELN;

  (* STALL is located in S.UTLS -- it waits for a <CR>. *)
  STALL;

  (* These files were open in ANALYZE when the error occurred. *)
  CLOSE(FILEITEMINFO,LOCK);
  CLOSE(DEST,LOCK);
  EXIT(ANALYZE);
END; (* error *)

(* ---------------------------------------------------------------------- *)

(* This procedure prints out the results of the infovalue calculations.  *)
(* This procedure is called from Procedure ANALYZE. *)
PROCEDURE RESULTS(COLUMN, ROW : INTEGER ; PRINTALL : BOOLEAN ;
                  T, INFOVAL, TOTAL : REAL);
BEGIN
  IF PRINTALL THEN BEGIN {1}

    (* print headings *)
    IF (COLUMN = 1) THEN BEGIN {2}
      (* DEST is declared and initialized in [redacted] *)
      WRITELN(DEST);
      WRITELN(DEST);
      PAGE(OUTPUT);

      (* DIRECTORY.TESTNAME is declared in
           and is initialized in Procedure [redacted] *)
      WRITELN(DEST,'SUBTEST : ',DIRECTORY.TESTNAME);
      WRITELN(DEST);

      (* T is global to procedure ANALYZE, and it is initialized
         also in procedure ANALYZE. *)
      WRITELN(DEST,'Theta : ',T);
      WRITELN(DEST);
      FOR HEADR := 1 TO 4 DO
        WRITE(DEST,' COL ITEM# INFOVALUE');

      (* screen automatically wraps around, printer will not. *)
      IF NOT (SELECT='2') THEN WRITELN(DEST);

      FOR HEADR := 1 TO 4 DO
        WRITE(DEST,' --- ----- ---------');
      WRITELN(DEST);
    END; {2}

    IF NOT (SELECT = '2') THEN WRITELN('Printing row : ',ROW);

    WRITE(DEST, COLUMN:3, ICODE:6, INFOVAL:11:7);
    IF (COLUMN MOD 4) = 0 THEN WRITELN(DEST);
    IF (COLUMN = ITEMS) THEN BEGIN {3}
      WRITELN(DEST);
      WRITELN(DEST,'Sum for THETA = ',T,' : ', TOTAL);
      WRITELN(DEST);
      (* stall if not writing to file *)
      (* Procedure STALL is located in S.UTLS *)
      IF NOT (SELECT = '3') THEN STALL;
    END;  (3)
  END   (1)
  ELSE

  (* Print only the sum of the infovalues for each row in infotable *)
  BEGIN (4)
    IF (COLUMN = 1) AND (ROW = 1) THEN BEGIN (5)
      PAGE(OUTPUT);
      WRITELN(DEST,'SUBTEST : ',DIRECTORY.TESTNAME);
      WRITELN(DEST);
    END;  (5)
    IF COLUMN = ITEMS THEN BEGIN (6)
      WRITELN(DEST,'THETA : ',T,'         SUM : ', TOTAL);

      IF ((ROW MOD 4) = 0) THEN WRITELN(DEST);

      (* 16 lines per screen *)
      IF (SELECT = '2') AND ((ROW MOD 16) = 0) THEN BEGIN (7)
        WRITELN(DEST);
        STALL;
        PAGE(OUTPUT);
        WRITELN(DEST,'SUBTEST : ',DIRECTORY.TESTNAME);
        WRITELN(DEST);
      END;  (7)
    END;  (6)

    (* stop after last item *)
    IF (SELECT = '2') AND (ROW = INFOROW) AND (COLUMN = ITEMS)
      THEN BEGIN (8)
             WRITELN(DEST);
             STALL;
           END;  (8)
  END;  (4)
END;  (* results *)

(* ----------------------------------------------------------------------- *)

(* print a histogram of the sums *)
(* This procedure is called by : Procedure ANALYZE *)
PROCEDURE GRAPH;
VAR N,SIZE : INTEGER;
    HALFSIZE : BOOLEAN;

    (* ................................................................... *)

    PROCEDURE GRAPHMENU;
    VAR OPTION : CHAR;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(20,0);
      WRITE('GRAPH  MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following options by entering its number.');
      GOTOXY(16,8);
      WRITE('1.  NO GRAPH');
      GOTOXY(16,9);
      WRITE('2.  DISPLAY GRAPH ON SCREEN');
      GOTOXY(16,10);
      WRITE('3.  DISPLAY GRAPH ON PRINTER');
      GOTOXY(16,11);
      WRITE('4.  WRITE GRAPH OUT TO FILE');
      GOTOXY(16,14);
      WRITE('Select choice # :');

      OPTION := GETCHAR(['1'..'4'],TRUE,FALSE,TRUE);
      CASE OPTION OF
       '1' : EXIT(GRAPH);
       '2' : REWRITE(DEST,'CONSOLE:');
       '3' : REWRITE(DEST,'PRINTER:');
```

```
            '4' : GETNEWFILE;
          END;  (* cases *)
        END;  (* graphmenu *)

        (* ---------------------------------------------------------------------- *)

        FUNCTION TOOLONG : BOOLEAN;
        VAR MAX : REAL;
            ROW : INTEGER;
        BEGIN
          MAX := 0.0;
          FOR ROW := 1 TO INFOROW DO
            BEGIN
              IF SUM[ROW] > MAX THEN
                MAX := SUM[ROW];
            END;
          IF MAX > 60.0 THEN
            TOOLONG := TRUE
          ELSE
            TOOLONG := FALSE;
        END;  (* toolong *)

    (* ............................................................................ *)

    BEGIN (* graph *)
      GRAPHMENU;
      PAGE(OUTPUT);
      WRITELN('READY TO PRINT GRAPH OF RESULTS');
      WRITELN('Be certain paper is alligned if using a printer.');
      WRITELN;
      STALL;
      PAGE(OUTPUT);
      WRITELN('WRITING GRAPH...');
      HALFSIZE := TOOLONG;

      (* print top heading and scale *)
      WRITELN(DEST,'SUBTEST : ',DIRECTORY.TESTNAME);
      WRITELN(DEST);
      WRITELN(DEST,'                       SUM OF ITEM INFORMATION VALUES OF BEST ',
              ITEMS,' ITEMS');
      WRITELN(DEST);
      WRITE(DEST,' THETA    SUM   ');
      FOR N := 1 TO 54 DO
        IF (N MOD 9) = 0 THEN
          IF HALFSIZE THEN
            WRITE(DEST, ( (N DIV 9) * 20) )
          ELSE
            WRITE(DEST, ( (N DIV 9) * 10) )
        ELSE
          WRITE(DEST,' ');
      WRITELN(DEST);
      WRITE(DEST,' -----    ---   ');
      FOR N := 1 TO 60 DO
        IF (N MOD 10) = 0 THEN
            WRITE(DEST,'|')
        ELSE
          WRITE(DEST,'-');
      WRITELN(DEST);

      (* print data and graph *)
      FOR ROW := 1 TO INFOROW DO
        BEGIN (1)
          T := TMIN + (DT * (ROW-1) );    (* T is THETA value *)
          WRITE(DEST,T:6:3,SUM[ROW]:8:3,' |');
          IF HALFSIZE THEN
            SIZE := TRUNC(SUM[ROW]/2)
          ELSE
            SIZE := TRUNC(SUM[ROW]);
          FOR N := 1 TO SIZE DO
            WRITE(DEST,'.');
          WRITELN(DEST);
        END;  (1)
      (* print scale at bottom of graph *)
      WRITE(DEST,'                 ');
      FOR N := 1 TO 60 DO
        IF (N MOD 10) = 0 THEN
          WRITE(DEST,'|')
        ELSE
          WRITE(DEST,'-');
      WRITELN(DEST);
      WRITE(DEST,'                   ');
      FOR N := 1 TO 54 DO
        IF (N MOD 9) = 0 THEN
          IF HALFSIZE THEN
            WRITE(DEST, ( (N DIV 9) * 20) )
          ELSE
            WRITE(DEST, ( (N DIV 9) * 10) )
        ELSE
          WRITE(DEST,' ');
      WRITELN(DEST);
      STALL;
      CLOSE(DEST,LOCK);
    END;  (* graph *)

(* ....................................................................... *)

BEGIN  (* Analyze *)

  (* Procedure LOADTEST is located in textfile S.MGR. *)
  LOADTEST('Analyze rows for which subtest? :');

  (* the boolean ESCPROC is global to S.MGR, and it is initialized in
     Procedure LOADTEST. *)
  IF ESCPROC THEN EXIT(ANALYZE);
  PAGE(OUTPUT);

  (* load the directory locations *)
  (* Procedure LOADINFO is located in textfile S.MGR *)
  LOADINFO(CURRINDEXRECNUM + MAXSUBTESTS + 1);

  (* PRINTALL is a boolean - False if only want to print Sums,
                            True if print infovalues & itemcodes. *)
  PRINTALL := PRINTDETAIL;

  REPEAT
    PAGE(OUTPUT);
    WRITELN('A sum of infovalues will be computed. Only the best ',INFOCOLUMN);
    WRITELN('items in each row of the infotable are available for use');
    WRITELN('in the computation.  You may select fewer.');
    WRITELN;
    WRITE('Enter the number of items per row to be summed over : ');
    READLN(ITEMS);
    (* INFOCOLUMN is a global constant *)
    IF (ITEMS > INFOCOLUMN) OR (ITEMS <= 1) THEN SQUAWK;
  UNTIL (ITEMS<=INFOCOLUMN) AND (ITEMS > 1);

  GETOUTPUTDEST;
  (* FILEITEMINFO is global to S.MGR -- DATANAME is a global constant *)
  RESET(FILEITEMINFO,DATANAME);

  PAGE(OUTPUT);
  (* INFOROW is a global constant *)
  FOR ROW := 1 TO INFOROW DO
  BEGIN (1)
    (* TMIN and DT are global constants *)
    T := TMIN + (DT * (ROW-1) );    (* T is THETA value *)
    SUM[ROW] := 0;

    FOR COLUMN := 1 TO ITEMS DO
    BEGIN (3)

      (* INFOTABLE is declared and initialized in Procedure INFOSETUP in
         textfile S.MGR *)
      DIRSLOT := INFOTABLE[COLUMN,ROW];
      IF (DIRSLOT<0) OR (DIRSLOT>300) THEN ERROR(1,DIRSLOT);
      (* DIRECTORY.ITEMCODE is global to S.MGR, and is initialized in
         Procedure LOADTEST in textfile S.MGR *)
      ICODE := DIRECTORY.ITEMCODE[DIRSLOT];
      IF ICODE < 0 THEN ERROR(2,ICODE);

      (* HASH is an integer function in textfile S.MGR *)
      DATAREC := HASH(DIRSLOT);

      SEEK(FILEITEMINFO,DATAREC);
      GET(FILEITEMINFO);

      (* ITEMINFO is global to S.MGR *)
      ITEMINFO := FILEITEMINFO^;
      A := ITEMINFO.A;
      B := ITEMINFO.B;
      C := ITEMINFO.C;
      INFOVAL := INFO(T,A,B,C);
      SUM[ROW] := SUM[ROW] + INFOVAL;

      (* print values calculated *)
      RESULTS(COLUMN, ROW, PRINTALL, T, INFOVAL, SUM[ROW]);
    END;  {3}
  END;  {1}
  CLOSE(FILEITEMINFO,LOCK);
  CLOSE(DEST,LOCK);
  GRAPH;
END;  (* analyze *)
```

GMGR.DIR:
Subdirectory - Graphics Management

```
(*$S+*)
(****************************************************************************)
(*                                                                          *)
(*      Textfile : G.MGR.TEXT                     Volume : TFILES           *)
(*      Codefile : G.MGR.CODE                     Volume : CATDATA          *)
(*                                                                          *)
(****************************************************************************)
(*  File last modified : Feb 28, 1983             NPROC                     *)
(****************************************************************************)


PROGRAM GRAPHICS_TEST_MANAGER;
USES CHAINSTUFF,
     PGRAF,
     APPLESTUFF,
     REALMODES,
     TRANSCEND;

CONST (* available screen dimensions *)
      NORTH = 191;
      SOUTH = 32;
      WEST = 0;
      EAST = 559;

      (* screen dimensions *)
      XSCREEN = 79;
      YSCREEN = 23;

      (* screen boundaries for question text *)
      (* lines 20 -23 reserved for system messages *)
      TOPMAX = 0;
      LEFTMAX = 0;
      BOTTOMMAX = 19;

      GOTOFLAG = 128;
      PAGEFLAG = 129;
      UNUSEDFLAG = 130;
      ENDITEM = 131;

      (* ascii values *)
      ETX = 3;        (* control-c *)
      BELL = 7;
      NUL = 0;
      LARROW = 8;
      RARROW = 21;
      RET = 13;
      UP = 11;        (* up arrow *)
      DOWN = 10;      (* down arrow *)
      ESC = 27;
      SPACE = 32;
      PAGEOUT = 16;  (* cntrl-p on apple II *)
      ASCIIOFFSET = 48; (* ascii zero *)

      INDEXNAME = 'CATDATA:TESTINDEX.DATA'; (* test directory *)


      (* slots available in directory *)
      MAXSUBTESTS = 20;
      (* maximum question pool per test *)
      MAXITEMPOOL = 300;

      VERSION = '[1.03]';

TYPE DIRDATA = PACKED RECORD  (* directory for tests *)
                  UNUSED : BOOLEAN;    (* tells if record occupied *)
                  TESTNAME : STRING;   (* name of subtest *)
                  (* subtest directory of question id codes *)
                  ITEMCODE : PACKED ARRAY
                             [0..MAXITEMPOOL]
                             OF INTEGER;
                END;


     SETOFCHAR = SET OF CHAR;

VAR X1, X2, Y1, Y2,
    XCOORD,
    YCOORD,
    HEADING,
    CURRINDEXRECNUM ,      (* subtest record  *)
    I,                     (* counter *)
    RSET,                  (* right border set by user *)
    LSET,                  (* left border set by user *)
    TSET,                  (* top border set by user *)
    BSET,                  (* bottom border set by user *)
    CURRBLOCK,             (* current block read/write *)
    CURRFREEPTR,           (* current byte location in current block *)
    OLDX,                  (* last x-location you were at on graphics screen *)
    OLDY,                  (* last y-location *)
    X,                     (* graphic screen coordinates *)
    Y : INTEGER;

    COMPRESSED,             (* true ==> graphics is compressed *)
    ESCPROC,                (* true ==> quit program *)
    FOREVER,                (* true ==> stay in loop *)
    ERASE,                  (* true ==> arrows will erase *)
    FASTCURSOR,             (* true ==> make piclette cursor go faster *)
    T,B,R,L,                (* true ==> borders were set by user *)
    PREVTRAVEL,             (* true ==> user moved cursor *)
    PREVBLACK,              (* true ==> last location was black *)
    NOMORE : BOOLEAN;       (* true ==> nothing left to do *)

    starttime,
    endtime,
    elasped : real;

    CH,
    ESCAPE,
    TLEFT,
    TRIGHT,
    TUP,
    TDOWN,
    COMMAND,
    OUTPUT : CHAR;

    FNAME,
    DESTNAME,                  (* where you what the picture krunched to *)
    SOURCENAME,                (* what fotofile you want to load *)
    VNAME,
    SUBTEST_CHAR : STRING;


    (* saves old screen colors where top and bottom borders lie *)
    TLINE,
    BLINE : PACKED ARRAY[WEST..EAST] OF BOOLEAN;

    (* saves old screen colors where left and right borders lie *)
    RLINE,
    LLINE : PACKED ARRAY[SOUTH..NORTH] OF BOOLEAN;


    (* graphics screen buffer *)
    (* gbuffer always reflects the original picture *)
    (* dotbuff reflects whats left of original picture after each *)
    (* stage of krunching *)
    GBUFFER,
    DOTBUFF : PACKED ARRAY[WEST..EAST,SOUTH..NORTH] OF BOOLEAN;


    (* compression data 4 blocks worth *)
    TRIX : RECORD CASE INTEGER OF
```

```
          1 : (ITEMBUF : ARRAY[0..1023] OF INTEGER);
          2 : (ASCIIBUF : PACKED ARRAY[0..2047] OF 0..139);
        END;

    (* directory record information *)
    DIRINFO : ARRAY[0..MAXSUBTESTS] OF RECORD

                                         (* record occupied *)
                                         NOTUSED : BOOLEAN;

                                         (* subtest name *)
                                         TNAME : STRING;

                                         (* # items in subtest *)
                                         ITEMCOUNT : INTEGER;
                                       END;

    screen : packed array[0..79,0..23] of char;

    DIRECTORY : DIRDATA;
    FILE_DIRECTORY : FILE OF DIR_DATA;

    (* graphics screen file *)
    GSCREEN : INTERACTIVE;

    (* compression file *)
    GTEXT : FILE;


PROCEDURE PAGE(DUMMY : CHAR);                                 FORWARD;
PROCEDURE SQUAWK;                                             FORWARD;
FUNCTION GETCHAR(OKSET : SETOFCHAR;
                 FLUSHQUEUE,ECHO,BEEP : BOOLEAN) : CHAR;      FORWARD;
PROCEDURE STALL;                                              FORWARD;
PROCEDURE BLANKLINES(START,COUNT,ENDCURSOR : INTEGER);        FORWARD;
FUNCTION TIME : REAL;                                         FORWARD;
PROCEDURE GGOTOXY(X,Y : INTEGER);                             FORWARD;
PROCEDURE GWRITESTR(GSTR : STRING);                           FORWARD;
PROCEDURE GWRITECHR(GCHR : CHAR);                             FORWARD;
PROCEDURE CLEARBOTTOM;                                        FORWARD;
PROCEDURE WRITEITEMBLOCK(WHICHBLOCK : INTEGER);               FORWARD;
PROCEDURE READITEMBLOCK(WHICHBLOCK : INTEGER);                FORWARD;
PROCEDURE TRAVEL(COORD : CHAR; VALUE : INTEGER);              FORWARD;
PROCEDURE MAINMENU;                                           FORWARD;


(*$I /TFILES/GMGR.DIR/G.1SUBRT.TEXT *)

(*$I /TFILES/GMGR.DIR/G.2SUBRT.TEXT *)

(*$I /TFILES/GMGR.DIR/G.UTL.TEXT *)     (* graphic  utilities *)

(*$I /TFILES/GMGR.DIR/G.SUBRT.TEXT *)


(* main line program *)
BEGIN   (* graphictestmgr *)
  FOREVER := TRUE;
  GETDIRINFO;
  REPEAT
    ESCPROC := FALSE;
    LOADTEST('Edit graphics for which subtest? : ');
    IF ESCPROC THEN
      FOREVER := FALSE
    ELSE
    BEGIN
      SUBTEST_CHAR := ' ';
      SUBTESTCHAR[1] := CHR(CURRINDEXRECNUM+65);
      INITIAL;
      FASTCURSOR := FALSE;
      ERASE := FALSE;
      PREVBLACK := FALSE;
      FIND;
      MAINMENU;

      REPEAT

        READ(KEYBOARD,CH);

        CASE CH OF
          '7' : PLOT('7',0);
          '9' : PLOT('9',0);
          '1' : PLOT('1',0);
          '3' : PLOT('3',0);
          '4' : PLOT('X',-1);
          '5' : PLOT('5',0);
          '6' : PLOT('X', 1);
          '8' : PLOT('Y', 1);
          '2' : PLOT('Y',-1);
          'D','d' : Draw;
          'C','c' : Characters;
          'H','h' : Find;
          'P','p' : Polygon;
          'R','r' : BEGIN
                      PAGE(OUTPUT);
                      TEXTON;
                      WRITE('Clear graphics screen ?   Y/N  : ');
                      IF GETCHAR(['Y','N','y','n'],TRUE,TRUE,TRUE) IN
                      ['Y','y'] THEN
                      BEGIN
                        GRAFIXON;
                        Remove;
                      END
                      ELSE
                        GRAFIXON;
                    END;
          'L','l' : Load;
          'S','s' : Save;
          'T','t' : ERASE :=  TRUE;
          'N','n' : FASTCURSOR := FALSE;
          'E','e' : ERASE := FALSE;
          'F','f' : FASTCURSOR := TRUE;
          'K','k' : KRUNCH;

          OTHERWISE
            IF (CH = TLEFT) THEN
              TRAVEL('X',-1)
            ELSE
            IF (CH = TRIGHT) THEN
              TRAVEL('X', 1)
            ELSE
            IF (CH = TUP) THEN
              TRAVEL('Y', 1)
            ELSE
            IF (CH = TDOWN) THEN
              TRAVEL('Y',-1);
          END;

        UNTIL CH IN ['Q','q'];
        CLOSE(GSCREEN,NORMAL);

    END;
  UNTIL NOT FOREVER;

  SETCHAIN('CATDATA:CATPROJECT');

END.
```

```
(*************************************************************)
(* FILE : G.UTL.TEXT                                         *)
(* File last modified :  Feb 25,1983                         *)
(*************************************************************)

(* clear the screen and put cursor at 0,0 *)
PROCEDURE PAGE;
BEGIN
  WRITE(CHR(28));
  GOTOXY(0,0);
END;  (* page *)

(**** rings the bell ****)
PROCEDURE SQUAWK;
BEGIN
  WRITE(CHR(BELL));
END;  (* squawk *)


(* read an acceptable character from the keyboard *)
FUNCTION GETCHAR;
VAR MASK : PACKED ARRAY[0..0] OF CHAR;
BEGIN
  IF FLUSHQUEUE THEN UNITCLEAR(2); (* flush buffer *)
  REPEAT
    UNITREAD(2,MASK,1);
    IF BEEP AND NOT (MASK[0] IN OKSET) THEN SQUAWK;

  UNTIL MASK[0] IN OKSET;
  IF ECHO AND (MASK[0] IN [CHR(32)..CHR(126)]) THEN
    WRITE(MASK[0]);
  GETCHAR := MASK[0];
END;  (* getchar *)


(**** display a message/wait for a keystroke ****)
PROCEDURE STALL;
VAR STALLCHAR : CHAR;
BEGIN
  WRITE('Press <RET> to continue ');
  STALLCHAR :=
    GETCHAR([CHR(RET),CHR(ESC)],TRUE,FALSE,TRUE);
  IF STALLCHAR = CHR(ESC) THEN EXIT(PROGRAM);
END;  (* stall *)


(**** blank out lines  ****)
PROCEDURE BLANKLINES;
VAR I : INTEGER;
BEGIN
  GOTOXY(0,START);
  FOR I := 1 TO (COUNT-1) DO
    WRITELN(' ' : 39);
  WRITE(' ':39);
  GOTOXY(0,ENDCURSOR);
END;   (* blanklines *)


(* returns the # seconds elasped since start of the day *)
FUNCTION TIME;
TYPE TIMERECORD = RECORD
                    S : PACKED ARRAY[1..12] OF CHAR;
                  END;
VAR HOUR,
    MINUTE,
    SECOND : REAL;

    T : TIMERECORD;
    TFILE : FILE OF TIMERECORD;
```

```
BEGIN

  RESET(TFILE,'.CLOCK');
  T := TFILE^;
  HOUR := ((ORD(T.S[6]) - 48.0) * 10.0) +
          (ORD(T.S[7]) - 48.0);
  MINUTE := ((ORD(T.S[8]) - 48.0) * 10.0) +
          (ORD(T.S[9]) - 48.0);
  SECOND := ((ORD(T.S[10]) - 48.0) * 10.0) +
          (ORD(T.S[11]) - 48.0);
  TIME := (HOUR * 3600.0) + (MINUTE * 60.0) + SECOND;

END;   (* time *)


(* does a gotoxy to the grafix screen using text coordinates *)
PROCEDURE GGOTOXY;
VAR XPOS,
    YPOS : INTEGER;
BEGIN
  XPOS := X * 7;
  YPOS := 191 - (Y * 8);
  MOVETO(XPOS,YPOS);
END;  (* ggotoxy *)


(* write a string to grafix *)
PROCEDURE GWRITESTR;
BEGIN
  UNITWRITE(3,GSTR[1],LENGTH(GSTR),0,12);
END;   (* gwritestr *)


(* does a write to graphics screen for char values *)
PROCEDURE GWRITECHR;
VAR C : STRING;
BEGIN
  C := ' ';
  C[1] := GCHR;
  UNITWRITE(3,C[1],1,0,12);
END;  (* gwritechr *)

{---------------------------------------------------------------------------}


(*  clear reserved spaced at bottom of graphics screen *)
PROCEDURE CLEARBOTTOM;
BEGIN
  VIEWPORT(0,559,0,SOUTH);
  FILLCOLOR(BLACK);
  FILLPORT;
  VIEWPORT(0,559,0,191);
  FILLCOLOR(WHITE);
END;  (* clearbottom *)


(* writes the graphics buffer to diskfile *)
PROCEDURE WRITEITEMBLOCK;
VAR BLOCKSTRANSFERRED : INTEGER;
    BADIO : BOOLEAN;
BEGIN
  BADIO := FALSE;
  RESET(GTEXT,DESTNAME);
  BLOCKSTRANSFERRED := BLOCKWRITE(GTEXT,TRIX.ITEMBUF,4,WHICHBLOCK);
  BADIO := ((BLOCKSTRANSFERRED < 4) OR (IORESULT <> 0));
  CLOSE(GTEXT,LOCK);
  IF BADIO THEN
  BEGIN
    WRITELN;
    WRITELN;
    WRITE(' Block ', WHICHBLOCK,' write error.');
    WRITELN;
    READLN;
    EXIT(PROGRAM);
  END;
END;  (* writeitemblock *)


(* reads a block from disk into the item ascii buffer *)
PROCEDURE READITEMBLOCK;
VAR BLOCKSTRANSFERRED : INTEGER;
    BADIO : BOOLEAN;
BEGIN
  BADIO := FALSE;
  RESET(GTEXT,DESTNAME);
  BLOCKSTRANSFERRED := BLOCKREAD(GTEXT,TRIX.ITEMBUF,4,WHICHBLOCK);
  BADIO := ((BLOCKSTRANSFERRED < 4) OR (IORESULT <> 0));
  CLOSE(GTEXT,LOCK);
  IF BADIO THEN
  BEGIN
    WRITELN;
    WRITELN;
    WRITE(' Block ', WHICHBLOCK,' write error.');
    WRITELN;
    READLN;
    EXIT(PROGRAM);
  END;
END;  (* readitemblock  *)


(* main menu of commands *)
PROCEDURE MAINMENU;
BEGIN
  CLEARBOTTOM;
  GGOTOXY(0,20);
  GWRITESTR('SUBTEST : ');
  GWRITESTR(DIRECTORY.TESTNAME);
  PENCOLOR(WHITE);
  FILLCOLOR(BLACK);
  GGOTOXY(0,21);
  GWRITESTR(
'N)ormal cursor    F)ast cursor    D)raw line    S)ave file        R)emove');
  GGOTOXY(0,22);
  GWRITESTR(
'<keypad #s draw>  P)olygon        H)ome         L)oad file        K)runch');
  GGOTOXY(0,23);
  GWRITESTR(
'<arrows to move>  C)haracters     E)rase off    T)urn erase on    Q)uit');

  PENCOLOR(BLACK);
  FILLCOLOR(WHITE);
  VIEWPORT(0,559,32,191);
  MOVETO(0,32);
END;    (* main menu *)


(* writes the directory information to record  *)
(* puts necessary file info in main memory     *)
PROCEDURE GETDIRINFO;
VAR I,K,ICOUNT : INTEGER;
BEGIN

  (* initialize the directory information *)
  FOR I := 0 TO MAXSUBTESTS DO
    DIRINFO[I].NOTUSED := TRUE;

  (* get the directory information *)
  I := 0;
  RESET(FILEDIRECTORY,INDEXNAME);
  REPEAT
    SEEK(FILEDIRECTORY,I);
    GET(FILEDIRECTORY);
    IF NOT (FILEDIRECTORY^.UNUSED) THEN
    BEGIN
```

```
      DIRECTORY := FILEDIRECTORY^;
      DIRINFO[I].NOTUSED := FALSE;
      DIRINFO[I].TNAME := DIRECTORY.TESTNAME;
      ICOUNT := 0;
      DIRINFO[I].ITEMCOUNT := ICOUNT;
    END;
    I := I + 1;
  UNTIL I > MAXSUBTESTS;
  CLOSE(FILEDIRECTORY,NORMAL);
END;  (* getdirinfo *)
```

```
(******************************************************************************)
(*  File last modified : May 6, 1983           NPROC                          *)
(******************************************************************************)
(* G.SUBRT *)


(* this procedure clears the entire screen             *)
(* this procedure is embedded in procedure edit        *)
PROCEDURE REMOVE;
BEGIN
  VIEWPORT(0,559,0,191);
  FILLCOLOR(BLACK);
  FILLPORT;

  VIEWPORT(0,559,32,191);  (* this initializes the drawing viewport   *)
  FILLCOLOR(WHITE);
  FILLPORT;
  MAINMENU;

END;       (* remove *)


(* this procedure clears the last four rows of the graphics screen *)
(* this procedure is embedded in procedure edit                    *)
PROCEDURE BOTTOM(BOTTOM_COLOR : SCREENCOLOR);
BEGIN
  VIEWPORT(0,559,0,32);
  FILLCOLOR(BOTTOM_COLOR);
  FILLPORT;
  VIEWPORT(0,559,32,191);

END;      (* bottom *)



(* this procedure is embedded in procedure edit                      *)
PROCEDURE INITIAL;
VAR
  ASCII : PACKED ARRAY[0..0] OF 0..255;
  I     : INTEGER;
  TESTNAME_LENGTH : INTEGER;

BEGIN
  INITGRAFIX;

  GRAFIXMODE(BW560,1);

  REMOVE;

  GRAFIXON;

  XCOORD := 0;
  YCOORD := 0;

  TLEFT := CHR(8);
  TRIGHT := CHR(21);
  TUP    := CHR(11);
  TDOWN  := CHR(10);

  ESCAPE := CHR(27);

  X1 := 0;
  X2 := 0;
  Y1 := 0;
  Y2 := 0;

  PREVTRAVEL := FALSE;

  REWRITE(GSCREEN,'.GRAFIX');
END;        (* initial *)
```

```
(* this procedure plots points on the screen                          *)
(* this procedure is embedded in procedure edit                       *)
PROCEDURE PLOT(COORD : CHAR; VALUE : INTEGER);
BEGIN
  XCOORD := XLOC;
  YCOORD := YLOC;

  IF ERASE THEN
  BEGIN
    PENCOLOR(WHITE);
    DOTAT(XCOORD,YCOORD);
    PENCOLOR(BLACK);
  END;


  IF COORD = 'X' THEN           (* compute new X coordinate *)
    XCOORD := XCOORD + VALUE
  ELSE
  IF COORD = 'Y' THEN           (* compute new Y coordinate *)
    YCOORD := YCOORD + VALUE
  ELSE
  IF COORD = '7' THEN           (* compute new upper left XY coordinate *)
    BEGIN
      XCOORD := XCOORD - 1;
      YCOORD := YCOORD + 1;
    END
  ELSE
  IF COORD = '9' THEN           (* compute new upper right XY coordinate *)
    BEGIN
      XCOORD := XCOORD + 1;
      YCOORD := YCOORD + 1;
    END
  ELSE
  IF COORD = '1' THEN           (* compute new lower left XY coordinate *)
    BEGIN
      XCOORD := XCOORD - 1;
      YCOORD := YCOORD - 1;
    END
  ELSE
  IF COORD = '3' THEN           (* compute new lower right XY coordinate *)
    BEGIN
      XCOORD := XCOORD + 1;
      YCOORD := YCOORD - 1;
    END;


  DOTAT(XCOORD,YCOORD);
  SOUND(1,1,63);
  PREV_TRAVEL := FALSE;

  IF NOT ERASE THEN
  BEGIN
    X1 := XLOC;
    Y1 := YLOC;
  END;

END;    (* plot *)



(* this procedure allows you to enter characters on the graphics screen *)
(* this procedure is embedded in procedure edit                         *)
PROCEDURE CHARACTERS;
VAR
  ASCII : PACKED ARRAY[0..8] OF 0..255;
  X,Y   : INTEGER;

BEGIN
  IF PREV_TRAVEL THEN
```

```
      BEGIN
        PENCOLOR(WHITE);
        DOTAT(XLOC,YLOC);
        PENCOLOR(BLACK);
      END;

    CLEARBOTTOM;
    GGOTOXY(0,20);
    GWRITESTR('SUBTEST : ');
    GWRITESTR(DIRECTORY.TESTNAME);
    PENCOLOR(WHITE);
    FILLCOLOR(BLACK);
    GGOTOXY(0,22);
    GWRITESTR('Enter characters.  Arrows move cursor');
    GGOTOXY(0,23);
    GWRITESTR('Press <ESC> to leave character mode.');

    PENCOLOR(BLACK);
    FILLCOLOR(WHITE);
    MOVETO(0,39);                   (* ALIGNMENT *)

    REPEAT
      (* CREATE CURSOR *)
      FILLCOLOR(BLACK);
      VIEWPORT(XLOC,XLOC+6,YLOC-7,YLOC);
      FILLPORT;

      UNITREAD(2,ASCII,1,,12);

      (* DELETE CURSOR *)
      FILLCOLOR(WHITE);
      FILLPORT;

      CASE ASCII[0] OF
         8 : MOVETO(XLOC - 7, YLOC);   (* left arrow *)
        21 : MOVETO(XLOC + 7, YLOC);   (* right arrow *)
        10 : MOVETO(XLOC,YLOC - 8);    (* down arrow *)
        11 : MOVETO(XLOC,YLOC + 8);    (* up arrow *)
        13 : MOVETO(0,YLOC - 8);       (* return key *)
        27 : ;
        OTHERWISE
             UNITWRITE(3,ASCII,1,,12);
      END;

    UNTIL ASCII[0] = 27;  (* escape *)

    VIEWPORT(0,559,32,191);

    PREVTRAVEL := FALSE;

    X := XLOC;
    Y := YLOC;

    MAINMENU;

    MOVETO(X,Y);

  END;   (* characters *)


  (* this procedure saves the graphics screen to the volume CATDATA: *)
  (* this procedure is embedded in procedure edit                    *)
  PROCEDURE SAVE;
  VAR
    GNAME,FNAME : STRING;
    A     : PACKED ARRAY[0..0] OF 29..29;
    SELECT : CHAR;

  BEGIN
    IF PREVTRAVEL THEN        (* eliminate travel dot *)
    BEGIN
    PENCOLOR(WHITE);
    DOTAT(XLOC,YLOC);
    PENCOLOR(BLACK);
  END;

  TEXTON;
  GOTOXY(0,0);
  WRITE(CHR(28));
  GOTOXY(18,0);
  WRITE('WRITE GRAPHICS MENU');
  GOTOXY(0,4);
  WRITE('Select one of the following options by entering its number.');
  GOTOXY(16,8);
  WRITE('1. QUIT');
  GOTOXY(16,9);
  WRITE('2. WRITE OUT AS SUBTEST QUESTION');
  GOTOXY(16,10);
  WRITE('3. WRITE OUT AS SUBTEST SAMPLE QUESTION');
  GOTOXY(16,11);
  WRITE('4. WRITE TO SPECIFIED FILENAME');
  GOTOXY(16,15);
  WRITE('Enter Choice # : ');
  SELECT := GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
  CASE SELECT OF
    '1' : BEGIN
            PAGE(OUTPUT);
            GRAFIXON;
            EXIT(SAVE);
          END;
    '2' : BEGIN
            BOTTOM(WHITE);
            PAGE(OUTPUT);
            WRITE('Enter the itemcode : ');
            READLN(QNAME);
            FNAME := CONCAT('/CATFOTO/',SUBTESTCHAR,'DIR/G',SUBTEST_CHAR,
                               'Q',QNAME,'.FOTO');
          END;
    '3' : BEGIN
            BOTTOM(WHITE);
            PAGE(OUTPUT);
            WRITE('Save as which sample question? <1..5> : ');
            SELECT := GETCHAR(['1'..'5'],TRUE,TRUE,TRUE);
            QNAME := ' ';
            QNAME[1] := SELECT;
            FNAME := CONCAT('/CATFOTO/',SUBTESTCHAR,'DIR/G',SUBTEST_CHAR,
                               'SQ',QNAME,'.FOTO');
          END;
    '4' : BEGIN
            BOTTOM(WHITE);
            PAGE(OUTPUT);
            WRITELN('Enter the filename you wish to save graphics in.');
            WRITELN('You may specify a volume and you must add a ".FOTO" ');
            WRITELN('at the end of the filename.');
            WRITELN;
            WRITE('Enter filename : ');
            READLN(FNAME);
          END;
  END;

  GSAVE(FNAME);

  WRITELN;
  WRITELN;
  WRITELN('Graphic pictured saved as ',FNAME);
  WRITELN;
  STALL;
  GRAFIXON;
  MAINMENU;
  WRITE(CHR(28));
  PREVTRAVEL := FALSE;
END;        (* save *)
```

```
(* this procedure loads a graphics file from CATDATA: *)
(* this procedure is embedded in procedure edit       *)
PROCEDURE LOAD;
VAR
  QNAME,FNAME : STRING;
  SELECT : CHAR;


  PROCEDURE QLOAD;
  BEGIN
    PAGE(OUTPUT);
    WRITE('Enter the itemcode : ');
    READLN(QNAME);
    FNAME := CONCAT('/CATFOTO/',SUBTESTCHAR,'DIR/G',SUBTEST_CHAR,
                    'Q',QNAME,'.FOTO');
  (*$I-*)
    RESET(GTEXT,FNAME);
  (*$I+*)

    IF IORESULT <> 0 THEN
    BEGIN
      FNAME := CONCAT('/CATFOTO/',SUBTESTCHAR,'DIR/G',SUBTEST_CHAR,
                      QNAME,'.DATA');
    (*$I-*)
      RESET(GTEXT,FNAME);
    (*$I+*)

      IF IORESULT <> 0 THEN
      BEGIN
        WRITELN;
        WRITELN('No such graphics for this question');
        WRITELN;
        STALL;
        GRAFIXON;
        PAGE(OUTPUT);
        CLOSE(GTEXT,NORMAL);
        EXIT(LOAD);
      END
      ELSE
      BEGIN
        COMPRESSED := TRUE;
        CLOSE(GTEXT,NORMAL);
      END;
    END
    ELSE
    BEGIN
      COMPRESSED := FALSE;
      CLOSE(GTEXT,NORMAL);
    END;

  END;   (* qload *)

  PROCEDURE SQLOAD;
  BEGIN
    PAGE(OUTPUT);
    WRITE('Read which sample question? <1..5> : ');
    SELECT := GETCHAR(['1'..'5'],TRUE,TRUE,TRUE);
    QNAME := ' ';
    QNAME[1] := SELECT;
    FNAME := CONCAT('/CATFOTO/',SUBTESTCHAR,'DIR/G',SUBTEST_CHAR,
                            'SQ',QNAME,'.FOTO');
  (*$I-*)
    RESET(GTEXT,FNAME);
  (*$I+*)

    IF IORESULT <> 0 THEN
    BEGIN
      FNAME := CONCAT('/CATFOTO/',SUBTESTCHAR,'DIR/G',SUBTEST_CHAR,
                      'SQ',QNAME,'.DATA');
    (*$I-*)
      RESET(GTEXT,FNAME);
  (*$I+*)

    IF IORESULT <> 0 THEN
    BEGIN
      WRITELN;
      WRITELN;
      WRITELN('No such graphics for this sample question');
      WRITELN;
      STALL;
      GRAFIXON;
      PAGE(OUTPUT);
      CLOSE(GTEXT,NORMAL);
      EXIT(LOAD);
    END
    ELSE
    BEGIN
      COMPRESSED := TRUE;
      CLOSE(GTEXT,NORMAL);
    END;
  END
  ELSE
  BEGIN
    COMPRESSED := FALSE;
    CLOSE(GTEXT,NORMAL);
  END;

END;   (* sqload *)


PROCEDURE FLOAD;
VAR ZNAME : STRING;
    LEAVE : BOOLEAN;
BEGIN
  PAGE(OUTPUT);
  WRITELN('Enter the filename you wish to read the graphics from.');
  WRITELN('You may specify a volume.');
  WRITELN;
  WRITE('Enter filename : ');
  READLN(ZNAME);
  IF POS('.FOTO',ZNAME) = 0 THEN
    FNAME  := CONCAT(ZNAME,'.FOTO');
(*$I-*)
  RESET(GTEXT,FNAME);
(*$I+*)

  LEAVE := FALSE;
  COMPRESSED := FALSE;

  IF IORESULT <> 0 THEN
  BEGIN

    COMPRESSED := TRUE;

    IF (POS('.FOTO',ZNAME) = 0) AND (POS('.DATA',ZNAME) = 0) THEN
    BEGIN
      FNAME  := CONCAT(ZNAME,'.DATA');

    (*$I-*)
      RESET(GTEXT,FNAME);
    (*$I+*)

      IF IORESULT <> 0 THEN
        LEAVE := TRUE;
    END
    ELSE
      LEAVE := TRUE;
  END;

  CLOSE(GTEXT,NORMAL);

  IF LEAVE THEN
  BEGIN
    WRITELN;
```

```
      WRITELN('No such graphics file.');
      WRITELN;
      STALL;
      GRAFIXON;
      PAGE(OUTPUT);
      EXIT(LOAD);
    END
  END;   (* fload  *)


BEGIN  (* load *)
  TEXTON;
  GOTOXY(0,0);
  WRITE(CHR(28));
  GOTOXY(18,0);
  WRITE('READ GRAPHICS MENU');
  GOTOXY(0,4);
  WRITE('Select one of the following options by entering its number.');
  GOTOXY(16,8);
  WRITE('1. QUIT');
  GOTOXY(16,9);
  WRITE('2. READ SUBTEST QUESTION');
  GOTOXY(16,10);
  WRITE('3. READ SUBTEST SAMPLE QUESTION');
  GOTOXY(16,11);
  WRITE('4. READ GRAPHICS FROM FILENAME');
  GOTOXY(16,15);
  WRITE('Enter Choice # : ');
  SELECT := GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
  CASE SELECT OF
    '1' : BEGIN
            PAGE(OUTPUT);
            GRAFIXON;
            EXIT(LOAD);
          END;
    '2' : QLOAD;
    '3' : SQLOAD;
    '4' : FLOAD;
  END;

  GRAFIXON;
  IF COMPRESSED THEN
  BEGIN
    DESTNAME := FNAME;
    DECODEGRAF;
  END
  ELSE
    GLOAD(FNAME);

  BOTTOM(BLACK);
  WRITE(CHR(28));
  MAINMENU;
  PREVTRAVEL := FALSE;
END;        (* load *)


 (* this procedure creates polygons                *)
 (* this procedure is embedded in procedure edit   *)
 PROCEDURE POLYGON;
 VAR
   I : INTEGER;
   SIZE, NUMSIDES : INTEGER;


   FUNCTION TANGLE : INTEGER;
   (* this function is embedded in procedure edit *)
   BEGIN
     IF HEADING < 0 THEN
       TANGLE := 360 + TANGLE
     ELSE
       TANGLE := HEADING;
   END;
```

322

```
   PROCEDURE MOVE(RELDISTANCE : INTEGER);
   (* this procedure is embedded in procedure edit  *)
   CONST
     RADCONST = 57.29578;

   BEGIN
     LINETO(ROUND(XLOC + 2 * RELDISTANCE * COS(TANGLE/RADCONST)),
            ROUND(YLOC + RELDISTANCE * SIN(TANGLE/RADCONST)));
   END;


   PROCEDURE TURN(RELANGLE : INTEGER);
   (* this procedure is embedded in procedure edit   *)
   BEGIN
     HEADING := (HEADING + RELANGLE) MOD 360;
   END;


 BEGIN
   HEADING := 0;
   TEXTON;
   WRITE('What size polygon ? ');
   READLN(SIZE);
   WRITE('How many sides ? ');
   READLN(NUMSIDES);
   GRAFIXON;

   FOR I := 1 TO NUMSIDES DO
     BEGIN
       MOVE(SIZE);
       TURN(360 DIV NUMSIDES);
     END;

   PREVTRAVEL := FALSE;

 END;    (* polygon *)


 (* this procedure draws a line from the last point you plotted to the *)
 (* position you are currently are on.                                 *)
 PROCEDURE DRAW;
 BEGIN
   X2 := XLOC;
   Y2 := YLOC;
   MOVETO(X1,Y1);
   LINETO(X2,Y2);
   X1 := XLOC;
   Y1 := YLOC;
   PREVTRAVEL := FALSE;
 END;    (* draw *)


 (* this procedure moves you to the lower left hand part *)
 (* of the drawing screen                                *)
 (* this procedure is embedded in procedure edit         *)
 PROCEDURE FIND;
 BEGIN
   IF PREVTRAVEL = TRUE THEN
     BEGIN
       PENCOLOR(WHITE);
       DOTAT(XLOC,YLOC);
       PENCOLOR(BLACK);
     END;
   MOVETO(0,32);
 END;  (* find *)
```

```
(* this procedure allows you to travel on the screen without plotting *)
(* a point                                                             *)
(* this procedure is embedded in procedure edit                        *)
PROCEDURE TRAVEL;
VAR MODVALUE : INTEGER;
BEGIN
  XCOORD := XLOC;
  YCOORD := YLOC;

  IF PREVTRAVEL THEN
  BEGIN
    IF (NOT PREVBLACK) THEN
    BEGIN
      PENCOLOR(WHITE);
      DOTAT(XLOC,YLOC);
      PENCOLOR(BLACK);
    END;
  END;

  MODVALUE := VALUE;
  IF FASTCURSOR THEN
    MODVALUE := MODVALUE * 8;

  IF COORD = 'X' THEN
    XCOORD := XCOORD + MODVALUE
  ELSE
    YCOORD := YCOORD + MODVALUE;

  IF XCOORD > EAST THEN
    XCOORD := EAST
  ELSE
    IF XCOORD < WEST THEN
      XCOORD := WEST;

  IF YCOORD > NORTH THEN
    YCOORD := NORTH
  ELSE
    IF YCOORD < SOUTH THEN
      YCOORD := SOUTH;

  MOVETO(XCOORD,YCOORD);

  IF XYCOLOR = 0 THEN
    PREVBLACK := TRUE
  ELSE
    PREVBLACK := FALSE;

  DOTAT(XCOORD,YCOORD);

  PREV_TRAVEL := TRUE;

END;     (* travel *)
```

```
(***********************************************************)
(* FILE : G.1SUBRT.TEXT                                    *)
(* File last modified :  Feb 25,1983                       *)
(***********************************************************)


(* reads the item text file and displays the graphics *)
SEGMENT PROCEDURE DECODEGRAF;
VAR X,
    Y,
    X1,Y1,
    CURRPTR,
    CURRBLK,
    DOTCNT : INTEGER;


    (* reads the item text file & displays item text *)
    PROCEDURE DECODEPRINT;
    VAR X,
        Y,
        BYTECNT,
        CHARCODE : INTEGER;



        (* return the next code in ascii file *)
        FUNCTION BUFCODE : INTEGER;
        BEGIN
          BUFCODE := TRIX.ASCIIBUF[CURRPTR];
          CURRPTR := CURRPTR + 1;
          IF CURRPTR > 2047 THEN
          (* end of block/get next block and reset byte ptr *)
            BEGIN (1)
              CURRBLK := CURRBLK + 4;
              READITEMBLOCK(CURRBLK);
              CURRPTR := 0;
            END;  (1)
        END;  (* bufcode *)


    BEGIN  (* decode print *)
      (* read bytes from the buffer *)
      REPEAT

        (* get char from block buffer *)
        CHARCODE := bufcode;

        CASE CHARCODE OF
          GOTOFLAG : BEGIN (1)    (* move cursor *)
                       (* next two bytes after flag are x,y coord *)
                       X := BUFCODE;
                       Y := BUFCODE;
                       BYTECNT := BUFCODE;
                       GGOTOXY(X,Y);
                     END;  (1)
          ENDITEM  : ;

        END;

        IF (CHARCODE >= 32) AND (CHARCODE <= 126) THEN
          GWRITECHR(CHR(CHARCODE));

      UNTIL CHARCODE = ENDITEM; (* until end flag hit *)
    END;  (* decodeprint *)



BEGIN  (* decode graf *)

  PENCOLOR(BLACK);
  GRAFIXON;
  FILLPORT;
  READITEMBLOCK(0);

  CURRBLK := 0;
  CURRPTR := 0;

  starttime := time;


  (* decode the xlines *)
  REPEAT
    IF CURRPTR > 1021 THEN
    (* end of block/get new block *)
    BEGIN
      CURRBLK := CURRBLK + 4;
      READITEMBLOCK(CURRBLK);
      CURRPTR := 0;
    END;
    X := TRIX.ITEMBUF[CURRPTR];
    IF X >= 0 THEN
    BEGIN
      Y := TRIX.ITEMBUF[CURRPTR + 1];
      MOVETO(X,Y);
      DOTCNT := TRIX.ITEMBUF[CURRPTR + 2];
      LINEREL(DOTCNT,0);
      CURRPTR := CURRPTR + 3;
    END;
  UNTIL X < 0;

  CURRPTR := CURRPTR + 1;

  (* decode the ylines *)
  REPEAT
    IF CURRPTR > 1021 THEN
    (* end of block/get new block *)
    BEGIN
      CURRBLK := CURRBLK + 4;
      READITEMBLOCK(CURRBLK);
      CURRPTR := 0;
    END;
    X := TRIX.ITEMBUF[CURRPTR];
    IF X >= 0 THEN
    BEGIN
      Y := TRIX.ITEMBUF[CURRPTR + 1];
      MOVETO(X,Y);
      DOTCNT := TRIX.ITEMBUF[CURRPTR + 2];
      LINEREL(0,DOTCNT);
      CURRPTR := CURRPTR + 3;
    END;
  UNTIL X < 0;

  CURRPTR := CURRPTR + 1;


  (* decode the diagonals *)
  REPEAT
    IF CURRPTR > 1020 THEN
    (* end of block/get new block *)
    BEGIN
      CURRBLK := CURRBLK + 4;
      READITEMBLOCK(CURRBLK);
      CURRPTR := 0;
    END;
    X := TRIX.ITEMBUF[CURRPTR];
    IF X >= 0 THEN
    BEGIN
      Y := TRIX.ITEMBUF[CURRPTR + 1];
      X1 := TRIX.ITEMBUF[CURRPTR + 2];
      Y1 := TRIX.ITEMBUF[CURRPTR + 3];
      MOVETO(X,Y);
      LINETO(X1,Y1);
      CURRPTR := CURRPTR + 4;
    END;
  UNTIL X < 0;
```

```
    CURRPTR := CURRPTR + 1;

    (* decode the dots *)
    REPEAT
      IF CURRPTR > 1022 THEN
      (* end of block/get new block *)
      BEGIN
        CURRBLK := CURRBLK + 4;
        READITEMBLOCK(CURRBLK);
        CURRPTR := 0;
      END;
      X := TRIX.ITEMBUF[CURRPTR];
      IF X >= 0 THEN
      BEGIN
        Y := TRIX.ITEMBUF[CURRPTR + 1];
        DOTAT(X,Y);
        CURRPTR := CURRPTR + 2;
      END;
    UNTIL X < 0;

    CURRPTR := CURRPTR + 1;
    CURRPTR := CURRPTR * 2;

    DECODEPRINT;

    endtime := time;

  END;    (* decodegraf *)


(* codes the graphics screen *)
SEGMENT PROCEDURE CODESCREEN;
 VAR X,Y,
     X0,Y0,
     DOTCOUNT,
     I : INTEGER;
     DONE : BOOLEAN;


     (* code into buffer lines > 1 dot *)
     PROCEDURE CODEXLINES;
     BEGIN

       FILLCOLOR(BLACK);
       PENCOLOR(WHITE);
       GGOTOXY(0,21);
       GWRITESTR('Coding horizontal lines                                   ');
       FILLCOLOR(WHITE);
       PENCOLOR(BLACK);

       (* start at line zero of screen *)
       Y := TSET;
       REPEAT

         (* point to first pixel on line *)
         X := LSET;

         REPEAT

           (* look for white space *)
           DONE := FALSE;
           REPEAT
             IF NOT (GBUFFER[X,Y]) THEN
               X := X + 1
             ELSE
               DONE := TRUE;
           UNTIL (X > RSET) OR (DONE);

           (* if the whole line was not blank *)
           IF X <= RSET THEN
           BEGIN
        (* save location to move to when decoding *)
        X0 := X;
        Y0 := Y;

        (* find out how many consecutive black dots *)
        DOTCOUNT := 0;
        DONE := FALSE;
        REPEAT
          DOTCOUNT := DOTCOUNT + 1;
          X := X + 1;
          IF X > RSET THEN
            DONE := TRUE
          ELSE
            IF NOT GBUFFER[X,Y] THEN
              DONE := TRUE;
        UNTIL DONE;

        (* save information only if more than one dot *)
        IF DOTCOUNT > 1 THEN
        BEGIN

          FOR I := 0 TO (DOTCOUNT-1) DO
          BEGIN
            DOTAT(X0 + I,Y0);
            DOTBUFF[X0 + I,Y0] := FALSE;
          END;

          (* dont seperate information over block boundaries *)
          IF CURRFREEPTR > 1021 THEN
          BEGIN
            WRITEITEMBLOCK(CURRBLOCK);
            CURRBLOCK := CURRBLOCK + 4;
            CURRFREEPTR := 0;
          END;

          TRIX.ITEMBUF[CURRFREEPTR] := X0;
          TRIX.ITEMBUF[CURRFREEPTR + 1] := Y0;
          TRIX.ITEMBUF[CURRFREEPTR + 2] := (DOTCOUNT - 1);
          CURRFREEPTR := CURRFREEPTR + 3;
        END;
      END;

    UNTIL X > RSET;

    (* point to next line *)
    Y := Y - 1;

  (* until all lines looked at *)
  UNTIL Y < BSET;

  (* dont seperate information over block boundaries *)
  IF CURRFREEPTR > 1021 THEN
  BEGIN
    WRITEITEMBLOCK(CURRBLOCK);
    CURRBLOCK := CURRBLOCK + 4;
    CURRFREEPTR := 0;
  END;

  (* flag the end of file *)
  TRIX.ITEMBUF[CURRFREEPTR] := -1;
  CURRFREEPTR := CURRFREEPTR + 1;

END;   (* codexlines *)


(* code into buffer lines > 1 dot *)
PROCEDURE CODEYLINES;
VAR DONTDO : BOOLEAN;


    PROCEDURE CHECKLINES;
    VAR BITSOFF,Y1 : INTEGER;
    BEGIN
```

```
       (* save information only if more than one dot *)
       IF DOTCOUNT < 2 THEN
         DONTDO := TRUE
       ELSE
       BEGIN

         (* skip all pixels already set by x lines *)
         (* checking from bottom to top *)
         DONE := FALSE;
         Y1 := Y0;
         REPEAT
           IF (DOTBUFF[X0,Y1]) THEN
             DONE := TRUE
           ELSE
             Y1 := Y1 + 1;
         UNTIL (Y1 >= (Y0 + DOTCOUNT)) OR (DONE);

         IF Y1 >= (Y0 + DOTCOUNT) THEN
           DONTDO := TRUE
         ELSE
         BEGIN

           (* set the new dot count *)
           DOTCOUNT := DOTCOUNT - (Y1 - Y0);

           (* dont do if only one dot *)
           IF DOTCOUNT < 2 THEN
             DONTDO := TRUE
           ELSE
           BEGIN

             (* save y marker for move to location *)
             Y0 := Y1;

             (* now look for dots already set by x lines from *)
             (* top to bottom of column *)
             Y1 := Y0 + DOTCOUNT - 1;
             REPEAT
               IF (DOTBUFF[X0,Y1]) THEN
                 DONE := TRUE
               ELSE
                 Y1 := Y1 - 1;
             UNTIL (Y1 <= Y0) OR (DONE);

             IF Y1 <= Y0  THEN
               DONTDO := TRUE
             ELSE
             BEGIN
               DOTCOUNT := (Y1 - Y0) + 1;

               (* check the ration of bits already on to bits off *)
               (* and draw line if bits off * 6 > # of bits to draw *)
               BITSOFF  := 0;
               FOR Y1 := Y0 TO (Y0 + DOTCOUNT - 1) DO
                 IF DOTBUFF[X0,Y1] THEN
                   BITSOFF := BITSOFF + 1;
               IF (BITSOFF * 6) < DOTCOUNT THEN
                 DONTDO := TRUE;

             END;
           END;
         END;
       END;
    END;   (* checklines *)

BEGIN

  PENCOLOR(WHITE);
  FILLCOLOR(BLACK);
  GGOTOXY(0,21);
  GWRITESTR('Coding vertical lines                                    ');
```

329

```
PENCOLOR(BLACK);
FILLCOLOR(WHITE);

(* start at line zero of screen *)
X := LSET;
REPEAT

  (* point to first pixel on line *)
  Y := BSET;

  REPEAT

    (* look for white space *)
    DONE := FALSE;
    REPEAT
      IF NOT (GBUFFER[X,Y]) THEN
        Y := Y + 1
      ELSE
        DONE := TRUE;
    UNTIL (Y > TSET) OR (DONE);

    (* if the whole line was not blank *)
    IF Y <= TSET THEN
    BEGIN

      (* save location to move to when decoding *)
      X0 := X;
      Y0 := Y;

      (* find out how many consecutive black dots *)
      DOTCOUNT := 0;
      DONE := FALSE;
      REPEAT
        DOTCOUNT := DOTCOUNT + 1;
        Y := Y + 1;
        IF Y > TSET THEN
          DONE := TRUE
        ELSE
          IF NOT GBUFFER[X,Y] THEN
            DONE := TRUE;
      UNTIL DONE;

      DONTDO := FALSE;
      CHECKLINES;

      IF NOT DONTDO THEN
      BEGIN


        FOR I := 0 TO (DOTCOUNT-1) DO
        BEGIN
          DOTAT(X0,Y0 + I);

          (* erase dots in buffer which will be drawn by line *)
          DOTBUFF[X0,Y0 + I] := FALSE;
        END;

        (* dont seperate information over block boundaries *)
        IF CURRFREEPTR > 1021 THEN
        BEGIN
          WRITEITEMBLOCK(CURRBLOCK);
          CURRBLOCK := CURRBLOCK + 4;
          CURRFREEPTR := 0;
        END;

        TRIX.ITEMBUF[CURRFREEPTR] := X0;
        TRIX.ITEMBUF[CURRFREEPTR + 1] := Y0;
        TRIX.ITEMBUF[CURRFREEPTR + 2] := (DOTCOUNT - 1);
        CURRFREEPTR := CURRFREEPTR + 3;
      END;
    END;

  UNTIL Y > TSET;
```

```
    (* point to next line *)
    X := X + 1;

  (* until all lines looked at *)
  UNTIL X > RSET;


  (* dont seperate information over block boundaries *)
  IF CURRFREEPTR > 1021 THEN
  BEGIN
    WRITEITEMBLOCK(CURRBLOCK);
    CURRBLOCK := CURRBLOCK + 4;
    CURRFREEPTR := 0;
  END;

  (* flag the end of file *)
  TRIX.ITEMBUF[CURRFREEPTR] := -1;
  CURRFREEPTR := CURRFREEPTR + 1;

END;   (* codeylines *)


(* code dots and diagonals *)
PROCEDURE CODEDIAGONALS;
VAR DOTCNT,
    XX,YY,
    X0,XF,
    Y0,YF : INTEGER;
    NOMOREDOTS,
    ZAPDOT : BOOLEAN;


    (* check diagonals  through  quadrants 1 and 3 *)
    PROCEDURE CHECK13;
    BEGIN
      ZAPDOT := FALSE;
      XX := X;
      YY := Y;
      DOTCNT := 0;
      NOMOREDOTS := FALSE;
      REPEAT
        IF DOTBUFF[XX,YY] THEN
        BEGIN
          XF := XX;
          YF := YY;
          XX := XX + 1;
          YY := YY + 1;
          DOTCNT := DOTCNT + 1;
        END
        ELSE
          NOMOREDOTS := TRUE;
      UNTIL (NOMOREDOTS) OR (XX > RSET) OR (YY > TSET);

      IF DOTCNT > 2 THEN
      BEGIN

        (* dont seperate information over block boundaries *)
        IF CURRFREEPTR > 1020 THEN
        BEGIN
          WRITEITEMBLOCK(CURRBLOCK);
          CURRBLOCK := CURRBLOCK + 4;
          CURRFREEPTR := 0;
        END;

        TRIX.ITEMBUF[CURRFREEPTR] := X;
        TRIX.ITEMBUF[CURRFREEPTR + 1] := Y;
        TRIX.ITEMBUF[CURRFREEPTR + 2] := XF;
        TRIX.ITEMBUF[CURRFREEPTR + 3] := YF;
        CURRFREEPTR := CURRFREEPTR + 4;

        X0 := X;
        Y0 := Y;
        REPEAT
          DOTAT(X0,Y0);
          DOTBUFF[X0,Y0] := FALSE;
          X0 := X0 + 1;
          Y0 := Y0 + 1;
        UNTIL X0 > XF;

        ZAPDOT := TRUE;
      END;
    END;  (* check13 *)


    (* check diagonals  through  quadrants 2 and 4 *)
    PROCEDURE CHECK24;
    BEGIN
      DOTBUFF[X,Y] := TRUE;
      XX := X;
      YY := Y;
      DOTCNT := 0;
      NOMOREDOTS := FALSE;
      REPEAT
        IF DOTBUFF[XX,YY] THEN
        BEGIN
          XF := XX;
          YF := YY;
          XX := XX + 1;
          YY := YY - 1;
          DOTCNT := DOTCNT + 1;
        END
        ELSE
          NOMOREDOTS := TRUE;
      UNTIL (NOMOREDOTS) OR (XX > RSET) OR (YY < BSET);

      IF DOTCNT > 3 THEN
      BEGIN

        (* dont seperate information over block boundaries *)
        IF CURRFREEPTR > 1020 THEN
        BEGIN
          WRITEITEMBLOCK(CURRBLOCK);
          CURRBLOCK := CURRBLOCK + 4;
          CURRFREEPTR := 0;
        END;

        TRIX.ITEMBUF[CURRFREEPTR] := X;
        TRIX.ITEMBUF[CURRFREEPTR + 1] := Y;
        TRIX.ITEMBUF[CURRFREEPTR + 2] := XF;
        TRIX.ITEMBUF[CURRFREEPTR + 3] := YF;
        CURRFREEPTR := CURRFREEPTR + 4;

        X0 := X;
        Y0 := Y;
        REPEAT;
          DOTAT(X0,Y0);
          DOTBUFF[X0,Y0] := FALSE;
          X0 := X0 + 1;
          Y0 := Y0 - 1;
        UNTIL X0 > XF;

      END;

      IF ZAPDOT THEN
        DOTBUFF[X,Y] := FALSE;

    END;  (* check24 *)

 BEGIN  (* codediagonals *)

   PENCOLOR(WHITE);
   FILLCOLOR(BLACK);
```

```
     GGOTOXY(0,21);
     GWRITESTR('Coding diagonal lines                      ');
     FILLCOLOR(WHITE);
     PENCOLOR(BLACK);


     ZAPDOT := FALSE;

     FOR X := LSET TO RSET DO
       FOR Y := BSET TO TSET DO
       BEGIN
         IF DOTBUFF[X,Y] THEN
         BEGIN
           CHECK13;
           CHECK24;
         END;
       END;

     (* dont seperate information over block boundaries *)
     IF CURRFREEPTR > 1020 THEN
     BEGIN
       WRITEITEMBLOCK(CURRBLOCK);
       CURRBLOCK := CURRBLOCK + 4;
       CURRFREEPTR := 0;
     END;

     (* flag the end of file *)
     TRIX.ITEMBUF[CURRFREEPTR] := -1;
     CURRFREEPTR := CURRFREEPTR + 1;
 END;  (* code diagonals *)


BEGIN    (* codescreen *)

 CURRBLOCK := 0;
 CURRFREEPTR := 0;

 PENCOLOR(BLACK);

 CODEXLINES;

 CODEYLINES;

 CODEDIAGONALS;

 FILLCOLOR(BLACK);
 PENCOLOR(WHITE);
 GGOTOXY(0,21);
 GWRITESTR('Coding dots                           ');
 FILLCOLOR(WHITE);
 PENCOLOR(BLACK);

 (* code the rest of the dots *)
 FOR X := LSET TO RSET DO
   FOR Y := BSET TO TSET DO
   BEGIN
     IF DOTBUFF[X,Y] THEN
     BEGIN

        (* dont seperate information over block boundaries *)
        IF CURRFREEPTR > 1022 THEN
        BEGIN
          WRITEITEMBLOCK(CURRBLOCK);
          CURRBLOCK := CURRBLOCK + 4;
          CURRFREEPTR := 0;
        END;

        TRIX.ITEMBUF[CURRFREEPTR] := X;
        TRIX.ITEMBUF[CURRFREEPTR + 1] := Y;
        CURRFREEPTR := CURRFREEPTR + 2;
        DOTAT(X,Y);
      END;
    END;
```

```
  (* dont seperate information over block boundaries *)
  IF CURRFREEPTR > 1022 THEN
  BEGIN
    WRITEITEMBLOCK(CURRBLOCK);
    CURRBLOCK := CURRBLOCK + 4;
    CURRFREEPTR := 0;
  END;

  (* flag the end of file *)
  TRIX.ITEMBUF[CURRFREEPTR] := -1;

  CURRFREEPTR := CURRFREEPTR + 1;

  (* adjust pointer for ascii buffer *)
  CURRFREEPTR := CURRFREEPTR * 2;

  (* dont seperate information over block boundaries *)
  IF CURRFREEPTR > 2047 THEN
  BEGIN
    WRITEITEMBLOCK(CURRBLOCK);
    CURRBLOCK := CURRBLOCK + 4;
    CURRFREEPTR := 0;
  END;
END;  (* codescreen *)


(* codes the screen array into compact block buffer and writes it to disk.  *)
SEGMENT PROCEDURE CODETEXT;
VAR X0,
    CHARCODE,X,Y,I,
    BLOCK,
    BYTE,
    BYTECOUNT : INTEGER;
DONE : BOOLEAN;

    {..................................................................}

    (* fill buffer with ascii & codes, write to disk  *)
    PROCEDURE FILLITEMBUFFER(BUFFCODE : INTEGER);
    BEGIN
      TRIX.ASCIIBUF[CURRFREEPTR] := BUFFCODE;
      CURRFREEPTR := CURRFREEPTR + 1;
      IF CURRFREEPTR = 2048 THEN
        BEGIN {1}
          WRITEITEMBLOCK(CURRBLOCK);
          CURRBLOCK := CURRBLOCK + 4;
          CURRFREEPTR := 0;
        END;  {1}
    END;  (* fillitembuffer *)

{......................................................................}

BEGIN  (* codescreen *)

 (* display message *)
 TEXTON;
 WRITE(CHR(28));
 GOTOXY(1;1);
 WRITE('Entering text');

 (* start at line 0 of screen buffer *)
 Y := 0;
 REPEAT

   (* point to first character on line *)
   X := 0;
   WRITE('.');

   (* look for leading blanks *)
   DONE := FALSE;
   REPEAT
  IF SCREEN[X,Y] = CHR(SPACE)
    THEN
        X := X + 1
    ELSE
        DONE := TRUE;
UNTIL  (X > XSCREEN) OR (DONE);

(* if the whole line was not blank  *)
IF X <= XSCREEN THEN
BEGIN (1)

  (* flag a gotoxy where first non-blank character begins *)
  FILLITEMBUFFER(GOTOFLAG);
  FILLITEMBUFFER(X);
  FILLITEMBUFFER(Y);
  BYTECOUNT := 0;

  (* figure out how many bytes on this line *)
  X0 := X;

  REPEAT
    I := X0;
    DONE := FALSE;

    REPEAT
      IF SCREEN[I,Y] = CHR(SPACE)
        THEN
            I := I + 1
        ELSE
            DONE := TRUE;
    UNTIL  (I > XSCREEN) OR (DONE);

    IF I < XSCREEN THEN
      BEGIN (2)
        BYTECOUNT := BYTECOUNT + 1;
        X0 := X0 + 1;
        DONE := FALSE;
      END   (2)
      ELSE
          DONE := TRUE;
  UNTIL (DONE) OR (X0 > XSCREEN);

  FILLITEMBUFFER(BYTECOUNT);

  REPEAT

    (* look for trailing blanks *)
    I := X;
    DONE := FALSE;

    REPEAT
      IF SCREEN[I,Y] = CHR(SPACE)
        THEN
            I := I + 1
        ELSE
            DONE := TRUE;
    UNTIL  (I > XSCREEN) OR (DONE);

    (* if the rest of the line was not all blanks *)
    IF I < XSCREEN THEN
      BEGIN (3)

        (* save the character in the block buffer *)
        FILLITEMBUFFER(ORD(SCREEN[X,Y]));

        (* set pointer to next character *)
        X := X + 1;

        (* set loop condition to process next character *)
        DONE := FALSE;
      END   (3)
      ELSE
```

```
             (* done with this line, go look at next line in screen buffer *)
             DONE := TRUE;

      (* until done with line or last character in line processed *)
     UNTIL (DONE) OR (X > XSCREEN);
   END;   (1)

   (* point to next line *)
   Y := Y + 1;

 (* until all lines looked at *)
 UNTIL Y >= YSCREEN;

 (* flag end of question text *)
 FILLITEMBUFFER(ENDITEM);

 (* write the last block of ascii to file *)
 WRITEITEMBLOCK(CURRBLOCK);

 GRAFIXON;

END;  (* codescreen *)

{--------------------------------------------------------------------------------}


(* fills an array representing the crt with char screen location may be       *)
(* specified this is the guts of the question text editor  *)
SEGMENT PROCEDURE FILLSCREENBUFFER(UPBOUND,RIGHTBOUND,
                                   LOBOUND,LEFTBOUND : INTEGER;
                                   FLUSHBUF : BOOLEAN);
VAR SCREENCHAR : CHAR;
    CHARACTERS,
    CONTROLCHAR : SETOFCHAR;
    SCREENBYTES,
    CHARCODE,
    X,
    Y,
    L : INTEGER;

BEGIN

  CHARACTERS := [CHR(32)..CHR(126)];

  (* set # of bytes in screen character buffer *)
  SCREENBYTES := (XSCREEN + 1) * (YSCREEN + 1);

  (* if wish to start with a blank buffer, new text *)
  (* clear screen *)
  FILLCHAR(SCREEN[0],SCREENBYTES,' ');

  (* put cursor in upper left hand corner *)
  X := LEFTBOUND;
  Y := UPBOUND;

  (* dont give multiple page option to graphics *)
  CONTROLCHAR := [CHR(UP),CHR(DOWN),CHR(LARROW),CHR(RARROW),
                  CHR(ETX),CHR(RET)];

  PENCOLOR(WHITE);
  FILLCOLOR(BLACK);
  GGOTOXY(0,20);
  FOR L := 1 TO RIGHTBOUND DO
    GWRITECHR('-');
  GGOTOXY(1,21);
  GWRITESTR('Enter krunched text. Use arrows to move cursor.');
  GGOTOXY(1,22);
  GWRITES[illegible]('<RET>:next line.');
  GGOTOX[illegible]23);
  GWRITESTR('<CTRL>-C to quit and save');
  PENCOLOR(BLACK);
  FILLCOLOR(WHITE);
```

```
(* fill up the screen character buffer until done *)
REPEAT

  (* monitor cursor location *)
  GGOTOXY(X,Y);
  FILLCOLOR(BLACK);
  VIEWPORT(XLOC,XLOC+6,YLOC-7,YLOC);
  FILLPORT;

  (* get a character from the keyboard *)
  SCREENCHAR := GETCHAR(CHARACTERS + CONTROLCHAR,TRUE,TRUE,TRUE);

  (* delete the graphics cursor *)
  FILLCOLOR(WHITE);
  FILLPORT;

  IF NOT (SCREENCHAR IN CHARACTERS) THEN
    GWRITECHR(SCREEN[X,Y]);

  (* get the ascii value *)
  CHARCODE := ORD(SCREENCHAR);

  (* check for cursor control characters *)
  CASE CHARCODE OF

    (* cursor moved up but not beyond set boundaries *)
    UP : IF Y <= UPBOUND
           THEN
               SQUAWK
           ELSE
               Y := Y - 1;

    (* cursor moved down but not beyond set boundaries *)
    DOWN : IF Y >= LOBOUND
             THEN
                 SQUAWK
             ELSE
                 Y := Y + 1;

    (* cursor mov[illegible]o left with auto wrap around *)
    LARROW : IF X <= LEFTBOUND THEN
             BEGIN (3)
               IF Y <= UPBOUND
                 THEN
                     SQUAWK
                 ELSE
                 BEGIN (4)
                   X := RIGHTBOUND;
                   Y := Y - 1;
                 END;  (4)
             END   (3)
             ELSE
                 X := X - 1;

    (* cursor moved to right with auto wraparound *)
    RARROW : IF X >= RIGHTBOUND THEN
               BEGIN (5)
                 IF Y >= LOBOUND
                   THEN
                       SQUAWK
                   ELSE
                   BEGIN (6)
                     Y := Y + 1;
                     X := LEFTBOUND;
                   END;  (6)
               END   (5)
               ELSE
                   X := X + 1;

    (* carriage return *)
    RET : IF Y >= LOBOUND
            THEN
                SQUAWK
            ELSE
            BEGIN {7}
              Y := Y + 1;
              X := LEFTBOUND;
            END;   {7}

    END;  (* cases *)

    (* if character typed was a visible character *)
    IF (CHARCODE >= 32) AND (CHARCODE <= 126) THEN
    BEGIN {10}

      (* if last character exceeded screen boundaries *)
      IF (X > RIGHTBOUND) AND (Y >= LOBOUND)
        THEN
            SQUAWK
        ELSE
        BEGIN {11}

          (* save the character in the screen buffer *)
          SCREEN[X,Y] := SCREENCHAR;

          (* write character to screen if graphics *)
          GWRITECHR(CHR(CHARCODE));

          (* move cursor over one *)
          X := X + 1;

          (* check auto wrap around *)
          IF X > RIGHTBOUND THEN
            BEGIN {12}
              IF Y < LOBOUND THEN
                BEGIN {13}
                  X := LEFTBOUND;
                  Y := Y + 1;
                END    {13}
                ELSE
                    X := X - 1;
            END;   {12}
        END;   {11}
    END;   {10}

  (* until control-c pressed *)
  UNTIL CHARCODE = ETX;

  (* restore the viewport *)
  VIEWPORT(0,559,32,191);

END;  (* fillscreenbuffer *)
```

```
(*********************************************************)
(* File : G.2SUBRT.TEXT                                 *)
(* Last modified : Feb 28, 1983                         *)
(*********************************************************)


(* draws the top border if user wishes to krunch subset of picture *)
SEGMENT PROCEDURE THANDLER;
BEGIN

  (* save the location cursor was at *)
  OLDX := XLOC;
  OLDY := YLOC;

  (* if cursor location is above the bottom border then it is ok *)
  IF YLOC > BSET THEN
  BEGIN

    (* erase old top set *)
    IF T THEN
    BEGIN
      FOR I := WEST TO EAST DO
      BEGIN
        IF TLINE[I] THEN
          PENCOLOR(BLACK)
        ELSE
          PENCOLOR(WHITE);
        DOTAT(I,TSET);
      END;
    END;

    (* draw the new top border and flag that border has been set *)
    T := TRUE;
    PENCOLOR(BLACK);
    PREVBLACK := TRUE;
    TSET := OLDY;
    FOR I := WEST TO EAST DO
    BEGIN
      MOVETO(I,TSET);

      (* save the original screenline where border overwrites it *)
      IF XYCOLOR = 0 THEN
        TLINE[I] := TRUE
      ELSE
        TLINE[I] := FALSE;

      IF (I >= LSET) AND (I <= RSET) THEN
        DOTAT(I,TSET);
    END;

    (* erase where original cursor was *)
    TLINE[OLDX] := FALSE;

    (* if the left border was set, adjust it so it meets with top border *)
    IF L THEN
    BEGIN
      MOVETO(LSET,SOUTH);
      PENCOLOR(WHITE);
      LINETO(LSET,BSET);
      PENCOLOR(BLACK);
      LINETO(LSET,TSET);
      PENCOLOR(WHITE);
      LINETO(LSET,NORTH);
    END;

    (* if the right border was set, adjust it *)
    IF R THEN
    BEGIN
      MOVETO(RSET,SOUTH);
      PENCOLOR(WHITE);
      LINETO(RSET,BSET);
      PENCOLOR(BLACK);
      LINETO(RSET,TSET);
      PENCOLOR(WHITE);
      LINETO(RSET,NORTH);
    END;

    (* restore cursor to original location *)
    PENCOLOR(BLACK);
    MOVETO(OLDX,OLDY);

  END;
END;  (* thandler *)


(* set the bottom border of area of graphics screen to compress *)
SEGMENT PROCEDURE BHANDLER;
BEGIN

  (* save current cursor location *)
  OLDX := XLOC;
  OLDY := YLOC;

  (* if cursor is below the top border then ok *)
  IF YLOC < TSET THEN
  BEGIN
    (* erase old bottom set if previously set *)
    IF B THEN
    BEGIN
      (* restore what screen was before *)
      FOR I := WEST TO EAST DO
      BEGIN
        IF BLINE[I] THEN
          PENCOLOR(BLACK)
        ELSE
          PENCOLOR(WHITE);
        DOTAT(I,BSET);
      END;
    END;

    B := TRUE;  (* flag that bottom was set *)

    (* draw new bottom border while saving original screen *)
    PREVBLACK := TRUE;
    PENCOLOR(BLACK);
    BSET := OLDY;
    FOR I := WEST TO EAST DO
    BEGIN
      MOVETO(I,BSET);
      IF XYCOLOR = 0 THEN
        BLINE[I] := TRUE
      ELSE
        BLINE[I] := FALSE;

      IF (I >= LSET) AND (I <= RSET) THEN
        DOTAT(I,BSET);
    END;
    BLINE[OLDX] := FALSE;

    (* adjust bottom border if left or right borders were set *)
    IF L THEN
    BEGIN
      MOVETO(LSET,SOUTH);
      PENCOLOR(WHITE);
      LINETO(LSET,BSET);
      PENCOLOR(BLACK);
      LINETO(LSET,TSET);
      PENCOLOR(WHITE);
      LINETO(LSET,NORTH);
    END;

    (* if right border was set *)
    IF R THEN
    BEGIN
      MOVETO(RSET,SOUTH);
      PENCOLOR(WHITE);
      LINETO(RSET,BSET);
      PENCOLOR(BLACK);
      LINETO(RSET,TSET);
      PENCOLOR(WHITE);
      LINETO(RSET,NORTH);
    END;

    PENCOLOR(BLACK);
    MOVETO(OLDX,OLDY);
  END;
END;  (* bhandler *)


(* set the left border of area to compress *)
SEGMENT PROCEDURE LHANDLER;
BEGIN

  (* save old cursor locations *)
  OLDX := XLOC;
  OLDY := YLOC;

  (* if current position is to the left of the right border then ok *)
  IF XLOC < RSET THEN
  BEGIN
    (* erase old left set *)
    IF L THEN
    BEGIN
      FOR I := SOUTH TO NORTH DO
      BEGIN
        IF LLINE[I] THEN
          PENCOLOR(BLACK)
        ELSE
          PENCOLOR(WHITE);
        DOTAT(LSET,I);
      END;
    END;

    (* draw the new border *)
    L := TRUE;
    PREVBLACK := TRUE;
    PENCOLOR(BLACK);
    LSET := OLDX;
    FOR I := SOUTH TO NORTH DO
    BEGIN
      MOVETO(LSET,I);
      IF XYCOLOR = 0 THEN
        LLINE[I] := TRUE
      ELSE
        LLINE[I] := FALSE;

      IF (I >= BSET) AND (I <= TSET) THEN
        DOTAT(LSET,I);
    END;
    LLINE[OLDY] := FALSE;

    (* adjust border if top or bottom was set *)
    IF T THEN
    BEGIN
      MOVETO(WEST,TSET);
      PENCOLOR(WHITE);
      LINETO(LSET,TSET);
      PENCOLOR(BLACK);
      LINETO(RSET,TSET);
      PENCOLOR(WHITE);
      LINETO(EAST,TSET);
    END;

    IF B THEN
    BEGIN
      MOVETO(WEST,BSET);
      PENCOLOR(WHITE);
      LINETO(LSET,BSET);
      PENCOLOR(BLACK);
      LINETO(RSET,BSET);
      PENCOLOR(WHITE);
      LINETO(EAST,BSET);
    END;
    PENCOLOR(BLACK);
    MOVETO(OLDX,OLDY);
  END;

END;  (* lhandler *)


(* set the right border of area to be compressed *)
SEGMENT PROCEDURE RHANDLER;
BEGIN

  OLDX := XLOC;
  OLDY := YLOC;
  IF XLOC > LSET THEN
  BEGIN
    (* erase old right   *)
    IF R THEN
    BEGIN
      FOR I := SOUTH TO NORTH DO
      BEGIN
        IF RLINE[I] THEN
          PENCOLOR(BLACK)
        ELSE
          PENCOLOR(WHITE);
        DOTAT(RSET,I);
      END;
    END;

    R := TRUE;
    PENCOLOR(BLACK);
    PREVBLACK := TRUE;
    RSET := OLDX;
    FOR I := SOUTH TO NORTH DO
    BEGIN
      MOVETO(RSET,I);
      IF XYCOLOR = 0 THEN
        RLINE[I] := TRUE
      ELSE
        RLINE[I] := FALSE;

      IF (I >= BSET) AND (I <= TSET) THEN
        DOTAT(RSET,I);
    END;
    RLINE[OLDY] := FALSE;

    IF T THEN
    BEGIN
      MOVETO(WEST,TSET);
      PENCOLOR(WHITE);
      LINETO(LSET,TSET);
      PENCOLOR(BLACK);
      LINETO(RSET,TSET);
      PENCOLOR(WHITE);
      LINETO(EAST,TSET);
    END;

    IF B THEN
    BEGIN
      MOVETO(WEST,BSET);
      PENCOLOR(WHITE);
      LINETO(LSET,BSET);
      PENCOLOR(BLACK);
      LINETO(RSET,BSET);
      PENCOLOR(WHITE);
      LINETO(EAST,BSET);
    END;
    PENCOLOR(BLACK);
    MOVETO(OLDX,OLDY);
  END;
END;  (* rhandler *)



(* krunch a fotofile *)
SEGMENT PROCEDURE KRUNCH;
VAR AGAIN : CHAR;


    (* get the filename to be compressed *)
    SEGMENT PROCEDURE GETCFILE;
    VAR QNAME : STRING;
        SELECT : CHAR;


        PROCEDURE CHECKIT;
        BEGIN
        (*$I-*)
          RESET(GTEXT,DESTNAME);
        (*$I+*)

          IF IORESULT = 0 THEN
          BEGIN
            WRITELN;
            WRITE('Destroy old ',DESTNAME,' ?  Y/N : ');
            IF GETCHAR(['Y','N','y','n'],TRUE,TRUE,TRUE) IN ['Y','y'] THEN
            BEGIN
              CLOSE(GTEXT,PURGE);
              REWRITE(GTEXT,DESTNAME);
            END
            ELSE
            BEGIN
              CLOSE(GTEXT,NORMAL);
              GRAFIXON;
              EXIT(KRUNCH);
            END;
          END
          ELSE
          BEGIN
          (*$I-*)
            REWRITE(GTEXT,DESTNAME);
          (*$I+*)

            IF IORESULT <> 0 THEN
            BEGIN
              WRITELN;
              WRITE('Cannot open ',DESTNAME,'    Press <RET> ');
              IF GETCHAR([CHR(RET)],TRUE,TRUE,TRUE) = CHR(RET) THEN
              BEGIN
                GRAFIXON;
                EXIT(KRUNCH);
              END;
            END;
          END;
        END;  (* checkit *)


    BEGIN
      IF PREVTRAVEL THEN       (* eliminate travel dot *)
      BEGIN
        PENCOLOR(WHITE);
        DOTAT(XLOC,YLOC);
        PENCOLOR(BLACK);
      END;

      TEXTON;
      GOTOXY(0,0);
      WRITE(CHR(28));
      GOTOXY(18,0);
      WRITE('KRUNCH GRAPHICS MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following options by entering its number.');
      GOTOXY(16,8);
      WRITE('1. QUIT');
      GOTOXY(16,9);
      WRITE('2. KRUNCH AS SUBTEST QUESTION');
      GOTOXY(16,10);
      WRITE('3. KRUNCH AS SUBTEST SAMPLE QUESTION');
      GOTOXY(16,11);
      WRITE('4. KRUNCH TO SPECIFIED FILENAME');
      GOTOXY(16,15);
      WRITE('Enter Choice # : ');
      SELECT := GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
      CASE SELECT OF
        '1' : BEGIN
                PAGE(OUTPUT);
                GRAFIXON;
                EXIT(KRUNCH);
              END;
        '2' : BEGIN
                PAGE(OUTPUT);
                WRITE('Enter the itemcode : ');
                READLN(QNAME);
                DESTNAME := CONCAT('/CATFOTO/',SUBTESTCHAR,'DIR/G',SUBTEST_CHAR,
                                   QNAME,'.DATA');
              END;
        '3' : BEGIN
                PAGE(OUTPUT);
                WRITE('Krunch as which sample question? <1..5> : ');
                SELECT := GETCHAR(['1'..'5'],TRUE,TRUE,TRUE);
                QNAME := ' ';
                QNAME[1] := SELECT;
                DESTNAME := CONCAT('/CATFOTO/',SUBTESTCHAR,'DIR/G',SUBTEST_CHAR,
                                   'SQ',QNAME,'.DATA');
              END;
        '4' : BEGIN
                PAGE(OUTPUT);
                WRITELN('Enter the filename you wish to krunch graphics to.');
                WRITELN('You may specify a volume.');
                WRITELN;
                WRITE('Enter filename : ');
                READLN(DESTNAME);
                IF POS('.DATA',DESTNAME) = 0 THEN
                  DESTNAME := CONCAT(DESTNAME,'.DATA');
              END;
      END;

      CHECKIT;

      CLOSE(GTEXT,LOCK);
    END;          (* getcname *)


BEGIN  (* compress *)

  PREVBLACK := FALSE;
  PREVTRAVEL := FALSE;
  T := FALSE;
  B := FALSE;
  R := FALSE;
  L := FALSE;
  TSET := NORTH;
  BSET := SOUTH;
  LSET := WEST;
  RSET := EAST;

  (* get a destination file for the compressed graphics *)
  GETCFILE;

  WRITEITEMBLOCK(0);
  GRAFIXON;
  CLEARBOTTOM;
  GGOTOXY(26,20);
  GWRITESTR('SET KRUNCH BORDERS');
  FILLCOLOR(BLACK);
  PENCOLOR(WHITE);
  GGOTOXY(0,21);
  GWRITESTR(
'T)op                  B)ottom                 L)eft                  R)ight');
  GGOTOXY(0,22);
  GWRITESTR(
'F)ast cursor          N)ormal cursor');
  GGOTOXY(0,23);
  GWRITESTR(
'<arrows move cursor>        <cntrl>-C  to accept');


  MOVETO(0,SOUTH);
  REPEAT

    CH := GETCHAR(['F','N','T','B','L','R',
                    'f','n','t','b','l','r',
                   CHR(LARROW),CHR(RARROW),CHR(UP),CHR(DOWN),
                   CHR(ETX)],TRUE,FALSE,TRUE);
    CASE CH OF

      (* set area to be compressed options *)
      'T','t' : THANDLER;
      'B','b' : BHANDLER;
      'L','l' : LHANDLER;
      'R','r' : RHANDLER;
      'F','f' : FASTCURSOR := TRUE;
      'N','n' : FASTCURSOR := FALSE;


    OTHERWISE

      (* just moves cursor, doesnt erase anything *)
      IF (CH = CHR(8)) THEN
        TRAVEL('X',-1)
      ELSE
        IF (CH = CHR(21)) THEN
          TRAVEL('X', 1)
        ELSE
          IF (CH = CHR(11)) THEN
            TRAVEL('Y', 1)
          ELSE
            IF (CH = CHR(10)) THEN
              TRAVEL('Y',-1);
    END;

  UNTIL ORD(CH) = ETX;

  PENCOLOR(BLACK);
  FILLCOLOR(WHITE);

  (* signal that crunching has begun *)
  CLEARBOTTOM;
  FILLCOLOR(BLACK);
  PENCOLOR(WHITE);
  GGOTOXY(0,21);
  GWRITESTR(
'                          Krunching Fotofile');
  PENCOLOR(BLACK);
  FILLCOLOR(WHITE);

  (* adjust the boundaries of area set by user to get all that is inside *)
  LSET := LSET + 1;
  TSET := TSET - 1;
  RSET := RSET - 1;
  BSET := BSET + 1;

  (* fill up dot buffer of what is exactly on the screen *)
  PENCOLOR(WHITE);
  FOR X := LSET TO RSET DO
  BEGIN
    FOR Y := BSET TO TSET DO
    BEGIN
      MOVETO(X,Y);
      GBUFFER[X,Y] := (XYCOLOR = 0);
      DOTAT(X,Y);
    END;
  END;
  DOTBUFF := GBUFFER;

  (* crunch the screen into new format *)
  CODESCREEN;

  (* signal crunching is done , give main menu back *)
  VIEWPORT(LSET,RSET,BSET,TSET);
  FILLCOLOR(BLACK);
  CLEARBOTTOM;
  PENCOLOR(WHITE);
  FILLCOLOR(BLACK);
  GGOTOXY(0,21);
  GWRITESTR('Krunching is completed.  Press <RET> to continue ');
  READLN;
  PENCOLOR(BLACK);
  FILLCOLOR(BLACK);
  CLEARBOTTOM;
  FILLSCREENBUFFER(0,78,20,0,TRUE);
  CODETEXT;

  DECODEGRAF;
  CLEARBOTTOM;
  GGOTOXY(0,20);
  FILLCOLOR(BLACK);
  PENCOLOR(WHITE);
  GWRITESTR('Press <RET> to continue.');
  FILLCOLOR(WHITE);
  PENCOLOR(BLACK);
  READLN;

  texton;
  write(chr(28));
  elasped := starttime - endtime;
  if elasped < 0 then
    elasped := - elasped;

  gotoxy(0,0);
  writeln('Time : ',elasped,' seconds');
  writeln;
  write('Press <RET>');
  if getchar([chr(ret)],true,true,true) = chr(ret) then
    grafixon;

  MAINMENU;
END;  (* compress *)



(* lists the tests in directory & loads *)
SEGMENT PROCEDURE LOADTEST(MESSAGE : STRING);
VAR Q,
    TESTNUM,
    RECNUM : INTEGER;
    OKTEST : BOOLEAN;
    TEXTCODE : CHAR;


    (* lists the directory test names to the screen *)
    PROCEDURE LISTTESTS;
    VAR I,J,K,ITEMCOUNT : INTEGER;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(16,0);
      WRITE('GRAPHICS MANAGER LIST OF SUBTESTS ',VERSION);
      I := 0;
      J := 0;
      GOTOXY(0,3);
      REPEAT
        IF NOT (DIRINFO[I].NOTUSED) THEN
        BEGIN
          J := J + 1;
          IF J <= 10 THEN
            GOTOXY(0,2+J)
          ELSE
            GOTOXY(40,2+J-10);
          WRITE(J,'.  ',DIRINFO[I].TNAME);
        END;
        I := I + 1;
      UNTIL I > MAXSUBTESTS;
    END;  (* listtests *)


BEGIN  (* loadtest *)
  TEXTON;
  OKTEST := FALSE;
  LISTTESTS;
  RESET(FILEDIRECTORY,INDEXNAME);
  REPEAT
    GOTOXY(0,15);
    WRITELN(
'INSTRUCTIONS : Enter choice #, then press <RET>.');
    WRITE(
'               To escape, press 0 then <RET>.');
    GOTOXY(0,18);
    WRITE(MESSAGE);
  (*$I-*)
    READLN(TESTNUM);
  (*$I+*)
    IF TESTNUM = 0 THEN
    BEGIN
      ESCPROC := TRUE;
      CLOSE(FILEDIRECTORY,LOCK);
      EXIT(LOADTEST);
    END;
    IF (TESTNUM < 0) OR (TESTNUM >  (MAXSUBTESTS+1)) THEN
    BEGIN
      WRITELN;
      WRITELN('Invalid test # : ',TESTNUM);
      SQUAWK;
      WRITELN;
      STALL;
    END
    ELSE
    BEGIN
      RECNUM := 0;
      Q := 0;
      REPEAT
        SEEK(FILEDIRECTORY,RECNUM);
        GET(FILEDIRECTORY);
        IF NOT (FILEDIRECTORY^.UNUSED) THEN
          Q := Q + 1;
        RECNUM := RECNUM + 1;
      UNTIL (Q = TESTNUM) OR (RECNUM > MAXSUBTESTS);
      IF Q = TESTNUM THEN
      BEGIN
        CURRINDEXRECNUM := RECNUM - 1;
        OKTEST := TRUE;
      END
      ELSE
      BEGIN
        WRITELN;
        WRITELN('No test loaded');
        WRITELN;
        STALL;
      END;
    END;
    IF NOT OKTEST THEN BLANKLINES(18,6,18);
  UNTIL OKTEST;
  SEEK(FILEDIRECTORY,CURRINDEXRECNUM);
  GET(FILEDIRECTORY);
  DIRECTORY := FILEDIRECTORY^;
  CLOSE(FILEDIRECTORY,LOCK);
  WRITE(CHR(28));
END;
```

DMGR.DIR:
Subdirectory - CAT diagnostic Program

```
(******************************************************************************)
(*                                                                            *)
(*      Textfile : D.MGR.TEXT                         Volume : TFILES         *)
(*      Codefile : D.MGR.CODE                         Volume : CATD[illegible]A        *)
(*                                                                            *)
(******************************************************************************)
(*  File last modified : Mar 11, 1983                 NPROC                   *)
(******************************************************************************)
(*      This program is a diagnostic procedure to look for possible bugs in   *)
(* the cat system files.  It checks all the infotables to see that the items  *)
(* exist in the database, and it also checks that all questions which have    *)
(* the graphics flag set indeed have a graphics foto on the volume Catfoto.   *)
(* Another option is a duplicate question search procedure which scans a      *)
(* specified subtest, and checks the question text to see if questions are    *)
(* similar or duplicate.                                                      *)
(* Other diagnostics will be added as needed.                                 *)
(*                                                                            *)
(******************************************************************************)

(*$S+*)
PROGRAM DIAGNOSTIC;
USES CHAINSTUFF;    (* allows to get back to catproject *)

CONST (* ascii values *)
        ETX = 3;        (* cntrl-c *)
        BELL = 7;
        NUL = 0;
        LARROW = 8;
        RARROW = 21;
        RET = 13;
        UP = 11;
        DOWN = 10;
        ESC = 27;
        SPACE = 32;
        ASCIIOFFSET = 48; (* ascii zero *)
        MAXLINEBUF = 79;   (* string buffer size *)

        (* test directory name *)
        INDEXNAME = 'CATDATA:TESTINDEX.DATA'; (* test directory *)
        DATANAME = 'CATDATA:ITEMPOOL.DATA';   (* Question directory *)
        TEXTNAME = 'QTEXT:ITEMTEXT.DATA';     (* Question ascii text *)

        (* slots available in directory *)
        MAXSUBTESTS = 20;

        (* maximum question pool per test *)
        MAXITEMPOOL = 300;

        (* maximum # of sample questions *)
        MAXSAMPLES = 5;

        UNITNUMPRINTER = 'PRINTER:';

        DEFAULTFILE = 'DIAGNOSTIC.TEXT';

        VERSION = '[1.03]';

        TABNAME = 'CATDATA:TABINFO.DATA';

        (* information table dimensions *)
        INFOROW = 36;
        INFOCOLUMN = 20;

TYPE DIRDATA = PACKED RECORD  (* directory for tests *)
                        UNUSED : BOOLEAN;
                        TESTNAME : STRING;
                        ITEMCODE : PACKED ARRAY
                                      [0..MAXITEMPOOL]
                                      OF INTEGER;
                      END;
```

```
     (* info table *)
     TABLE = ARRAY[1..INFOCOLUMN,1..INFOROW] OF INTEGER;

     (* type of question response *)
     SEVENTYPE = PACKED ARRAY[1..7] OF CHAR;

     (* Different types of ways to answer a question *)
     ITEMRESPONSES = (CHARVALUE,        (* normal multiple choice *)
                      INTVALUE,         (* Integer value as answer *)
                      SEVENCHR);        (* seven characters saved as answer *)

     (* question ptrs/data , information for each question *)
     ITEMDATA = PACKED RECORD

                         (* flags if graphics item *)
                         GRAPHICS : BOOLEAN;

                         (* valid response ranges for multiple choice *)
                         LOWANSWER,
                         HIGHANSWER : CHAR;

                         (* block # in file where text starts *)
                         ITEMBLOCK,

                         (* byte # in block where text starts *)
                         ITEMPTR,

                         (* # of answers if multiple question screen *)
                         ANSWERCOUNT  : INTEGER;

                         (* information parameters for bayesian strategy *)
                         A,B,C,

                         (* currently unused *)
                         PROPCORRECT,
                         POINTBISERIAL,
                         YOPT,
                         XOPT,
                         DUMMY1,  (* used to flag compressed graphics if 1.0 *)
                         DUMMY2,
                         DUMMY3 : REAL;

                         (* correct answer to question *)
                         CASE ATYPE : ITEMRESPONSES OF
                           CHARVALUE : (ANSWER : CHAR);
                           INTVALUE  : (INTANSWER : INTEGER);
                           SEVENCHR  : (CHRANSWER : SEVENTYPE);
                       END;

     SETOFCHAR = SET OF CHAR;

VAR LETTERS,DIGITS,CHARACTERS : SET OF CHAR;
    output,
    COMMAND : CHAR;

    ESCPROC : BOOLEAN;  (* true ==> leave duplicate question search *)

    (* string character buffer *)
    LINEBUF : PACKED ARRAY[0..MAXLINEBUF] OF CHAR;

    CURRINDEXRECNUM : INTEGER;  (* record # of file directory *)

    (* test directory *)
    DIRECTORY : DIRDATA;
    FILEDIRECTORY : FILE OF DIRDATA;

    (* test question ptrs/data *)
    ITEMINFO : ITEMDATA;
    FILEITEMINFO : FILE OF ITEMDATA;

    (* infotable *)
    INFOTABLE : TABLE;
    INFOFILE  : FILE OF TABLE;
    (* question ascii file *)
    ITEMTEXT : FILE;

    (* output file for test listings *)
    OUTFILE,
    DEST : TEXT;

PROCEDURE PAGE(DUMMY : CHAR);                                    FORWARD;
PROCEDURE SQUAWK;                                                FORWARD;
PROCEDURE BLANKLINES(START,COUNT,ENDCURSOR : INTEGER);           FORWARD;
FUNCTION GETCHAR(OKSET : SETOFCHAR;
                 FLUSHQUEUE,ECHO,BEEP : BOOLEAN) : CHAR;         FORWARD;
PROCEDURE STALL;                                                 FORWARD;
FUNCTION SLOT(CODE : INTEGER) : INTEGER;                         FORWARD;
PROCEDURE MENU;                                                  FORWARD;
PROCEDURE LOADINFO(RECNUM : INTEGER);                            FORWARD;
FUNCTION HASH(KEY : INTEGER) : INTEGER;                          FORWARD;
PROCEDURE  GETNEWFILE;                                           FORWARD;
PROCEDURE LOADTEST(MESSAGE : STRING);                          FORWARD;


(*$I /TFILES/DMGR.DIR/D.SEARCH.TEXT *)   (* looks for duplicate questions *)

(*$I /TFILES/DMGR.DIR/D.INFOTAB.TEXT *)  (* check the infotables *)

(*$I /TFILES/DMGR.DIR/D.GRAFIX.TEXT *)   (* check the grafix files *)

(*$I /TFILES/DMGR.DIR/D.UTL.TEXT *)      (* utilities *)


(* main program *)
BEGIN
  DIGITS := ['0'..'9'];
  LETTERS := ['A'..'Z','a'..'z'];
  CHARACTERS := [CHR(32)..CHR(126)];
  FILLCHAR(LINEBUF[0],MAXLINEBUF,' ');

  REPEAT
    ESCPROC := FALSE;
    MENU;
    COMMAND := GETCHAR(['1'..'4'],TRUE,FALSE,TRUE);
    CASE COMMAND OF
      '1' : ;
      '2' : CHECKINFOTABS;
      '3' : CHECKGFILES;
      '4' : SEARCHTEXT;
    END;  (* cases *)
  UNTIL COMMAND = '1';

  PAGE(OUTPUT);
  GOTOXY(18,10);
  WRITE('Loading Catproject driver');
  SETCHAIN('CATDATA:CATPROJECT');
END. (* diagnostic *)
```

```
(********************************************************************)
(* FILE : D.UTL.TEXT                  "Include file for D.MGR"      *)
(* File last modified : Mar 11, 1983                                *)
(********************************************************************)

(* This procedure is called by most of the other procedures.  It clears *)
(* the console screen. *)
PROCEDURE PAGE;
BEGIN
  WRITE(CHR(28));
  GOTOXY(0,0);
END;


(**** rings the bell ****)
PROCEDURE SQUAWK;
BEGIN
  WRITE(CHR(BELL));
END;  (* squawk *)

(* -------------------------------------------------------------------------- *)

(**** blank out lines  ****)
PROCEDURE BLANKLINES;
VAR I : INTEGER;
BEGIN
  GOTOXY(0,START);
  FOR I := 1 TO (COUNT-1) DO
    WRITELN(' ' : 39);
  WRITE(' ':39);
  GOTOXY(0,ENDCURSOR);
END;   (* blanklines *)

(* -------------------------------------------------------------------------- *)

(* read an acceptable character from the keyboard *)
FUNCTION GETCHAR;
VAR MASK : PACKED ARRAY[0..0] OF CHAR;
BEGIN
  IF FLUSHQUEUE THEN UNITCLEAR(2); (* flush buffer *)
  REPEAT
    UNITREAD(2,MASK,1);
    IF BEEP AND NOT (MASK[0] IN OKSET) THEN SQUAWK;

  UNTIL MASK[0] IN OKSET;
  IF ECHO AND (MASK[0] IN [CHR(32)..CHR(126)]) THEN
    WRITE(MASK[0]);
  GETCHAR := MASK[0];
END;  (* getchar *)

(* -------------------------------------------------------------------------- *)

(**** display a message/wait for a keystroke ****)
PROCEDURE STALL;
VAR STALLCHAR : CHAR;
BEGIN
  WRITE('Press <RET> to continue ');
  STALLCHAR :=
    GETCHAR([CHR(RET),CHR(ESC)],TRUE,FALSE,TRUE);
  IF STALLCHAR = CHR(ESC) THEN EXIT(PROGRAM);
END;  (* stall *)

(* -------------------------------------------------------------------------- *)


(* Given a question code, this function returns the location *)
(* of the question's data, & text pointers.                  *)
(* This function is called by: Procedure ? *)

FUNCTION SLOT;
VAR INDEX : INTEGER;
    FOUND : BOOLEAN;
    (* All other variables are global to Program STRATEGY *)

BEGIN  (* slot *)
  INDEX := MAXSAMPLES + 1;
  FOUND := FALSE;

  REPEAT
    IF DIRECTORY.ITEMCODE[INDEX] = CODE
       THEN
           FOUND := TRUE
       ELSE
           INDEX := INDEX + 1;
   UNTIL (INDEX > MAXITEMPOOL) OR (FOUND);

   IF FOUND
     THEN
         SLOT := INDEX
     ELSE
         SLOT := -1;
 END;  (* slot *)

 (* ------------------------------------------------------------------- *)


(* Show command level selections *)
(* This procedure is called by Program STRATEGY main routine. *)

 PROCEDURE MENU;
 BEGIN
   PAGE(OUTPUT);
   GOTOXY(20,0);
   WRITE('CAT SYSTEM DIAGNOSTIC MENU ',VERSION);
   GOTOXY(0,4);
   WRITE('Select one of the following options by entering its number.');
   GOTOXY(18,8);
   WRITE('1.  QUIT');
   GOTOXY(18,9);
   WRITE('2.  CHECK INFOTABLES');
   GOTOXY(18,10);
   WRITE('3.  CHECK GRAPHICS FILES');
   GOTOXY(18,11);
   WRITE('4.  LOOK FOR DUPLICATE/SIMILAR QUESTIONS');
   GOTOXY(18,15);
   WRITE('Enter Choice # : ');
 END; (* menu *)

(* ------------------------------------------------------------------- *)


(* This procedure loads the information table for a subtest. *)
(* This procedure is called by: Procedure *)

PROCEDURE LOADINFO;
(* INFOFILE are declared in Procedure INFOSETUP. INFOTABLE is an     *)
(* array.  *)

BEGIN
  RESET(INFOFILE,TABNAME);
  SEEK(INFOFILE,RECNUM);
  GET(INFOFILE);
  INFOTABLE := INFOFILE^;
  CLOSE(INFOFILE,NORMAL);
END;  (* loadinfo *)


FUNCTION HASH;
BEGIN
  HASH := (CURRINDEXRECNUM * MAXITEMPOOL)
            + KEY + CURRINDEXRECNUM;
END; (* Hash *)
```

```
(**** open a new text file ****)
PROCEDURE  GETNEWFILE;
VAR FILENAME : STRING;
    ERRNUM : INTEGER;

    (* ................................................................ *)

    (**** get a legal filename ****)
    FUNCTION NAMEOK : BOOLEAN;
    VAR I : INTEGER;
    BEGIN
      IF FILENAME = '' THEN
      BEGIN {1}
        FILENAME := DEFAULTFILE;
        GOTOXY(44,0);
        WRITE(FILENAME);
      END   {1}
      ELSE
        IF FILENAME[1] = CHR(esc) THEN EXIT(PROGRAM);
      IF (POS('.TEXT',FILENAME) <> (LENGTH(FILENAME) - 4))
         OR (LENGTH(FILENAME) < 6 ) THEN
           FILENAME := CONCAT(FILENAME,'.TEXT');
    (*$I-*)
      RESET(DEST,FILENAME);
    (*$I+*)
      IF IORESULT = 0 THEN
      BEGIN {2}
        WRITELN;
        WRITELN;
        WRITE('Destroy old ',FILENAME,'?   Press ''N'' or ''Y'' ');
        IF GETCHAR(['y','n','Y','N'],TRUE,TRUE,TRUE) IN ['Y','y'] THEN
        BEGIN {3}
          CLOSE(DEST,PURGE);
          REWRITE(DEST,FILENAME);
          NAMEOK := TRUE;
        END   {3}
        ELSE
          NAMEOK := FALSE;
      END   {2}
      ELSE
      BEGIN {4}
      (*$I-*)
        REWRITE(DEST,FILENAME);
      (*$I+*)
        ERRNUM := IORESULT;
        IF IORESULT <> 0 THEN
        BEGIN {5}
          WRITELN;
          WRITELN;
          WRITELN('Cannot open ',FILENAME,' Io error #',ERRNUM);
          NAMEOK := FALSE;
        END   {5}
        ELSE
          NAMEOK := TRUE;
      END;  {4}
    END;  (* nameok *)

(* ................................................................ *)

BEGIN  (* getnewfile  *)
  REPEAT
    PAGE(OUTPUT);
    WRITE('Enter output file name, then press <RET>  : ');
    READLN(FILENAME);
  UNTIL NAMEOK;
END;


(* lists the tests in directory & loads *)
PROCEDURE LOADTEST;
VAR I,J,Q,
    TESTNUM,
    RECNUM : INTEGER;
    OKTEST : BOOLEAN;
    TEXTCODE : CHAR;

BEGIN  (* load test *)
  OKTEST := FALSE;
  PAGE(OUTPUT);
  GOTOXY(10,0);
  WRITE('DUPLICATE QUESTION SEARCH (LIST OF SUBTESTS)');
  GOTOXY(0,3);

  (* get the directory information *)
  I := 0;
  J := 0;
  RESET(FILEDIRECTORY,INDEXNAME);
  REPEAT
    SEEK(FILEDIRECTORY,I);
    GET(FILEDIRECTORY);
    IF NOT (FILEDIRECTORY^.UNUSED) THEN
    BEGIN (1)
      J := J + 1;
      IF J <= 10
        THEN
            GOTOXY(0,2+J)
        ELSE
            GOTOXY(40,2+J-10);
      WRITELN(J,'.  ',FILEDIRECTORY^.TESTNAME);
    END;  (1)
    I := I + 1;
  UNTIL I > MAXSUBTESTS;

  REPEAT
    GOTOXY(0,15);
    WRITELN('INSTRUCTIONS : Enter choice #, then press <RET>.');
    WRITE('                 To escape, press 0 then <RET>.');
    GOTOXY(0,18);
    WRITE(MESSAGE);
  (*$I-*)
    READLN(TESTNUM);
  (*$I+*)

    IF TESTNUM = 0
      THEN
          BEGIN (1)
            ESCPROC := TRUE;
            CLOSE(FILEDIRECTORY,LOCK);
            EXIT(LOADTEST);
          END;  (1)

    IF (TESTNUM < 0) OR (TESTNUM >  (MAXSUBTESTS+1))
      THEN
          BEGIN (2)
            WRITELN;
            WRITELN('Invalid test # : ',TESTNUM);
            SQUAWK;
            WRITELN;
            STALL;
          END   (2)
      ELSE
          BEGIN (3)
            RECNUM := 0;
            Q := 0;

            REPEAT
              SEEK(FILEDIRECTORY,RECNUM);
              GET(FILEDIRECTORY);
              IF NOT (FILEDIRECTORY^.UNUSED) THEN Q := Q + 1;
              RECNUM := RECNUM + 1;
            UNTIL (Q = TESTNUM) OR (RECNUM > MAXSUBTESTS);

            IF Q = TESTNUM
              THEN
                  BEGIN (4)
```

```
                    CURRINDEXRECNUM := RECNUM - 1;
                    OKTEST := TRUE;
                  END    {4}
              ELSE
                  BEGIN {5}
                    WRITELN;
                    WRITELN('No test loaded');
                    WRITELN;
                    STALL;
                  END;   {5}
          END;   {3}
    IF NOT OKTEST THEN BLANKLINES(18,6,18);
  UNTIL OKTEST;

  SEEK(FILEDIRECTORY,CURRINDEXRECNUM);
  GET(FILEDIRECTORY);
  DIRECTORY := FILEDIRECTORY^;
  CLOSE(FILEDIRECTORY,LOCK);
END;  (* load test *)
```

```
(******************************************************************************)
(* Textfile : D.SEARCH.TEXT                                                   *)
(* File last modified : Mar 11, 1983        "Include file for D.MGR"          *)
(******************************************************************************)
(*                                                                            *)
(*  This procedure scans through the CAT system ascii files and searches for  *)
(*  questions which are identical or similar.  First, a subtest is chosen     *)
(*  to scan, then two pieces of data are required : 1. The number of key      *)
(*  words to extract from each question, and 2. A minimum criterion for       *)
(*  determining if a question is similar.                                     *)
(*                                                                            *)
(*    The algorithm uses these two pieces of data in the following manner.    *)
(*                                                                            *)
(*  A procedure scans each question, extracting the X longest words from each *)
(*  question.  This X value is the number of key words desired by the user.   *)
(*  These words are normalized into uppercase, the first 8 characters saved,  *)
(*  and stored in a packed array.                                             *)
(*                                                                            *)
(*  Each question's key words is compared to all previous scanned questions'  *)
(*  key words and the number of matches is noted.                             *)
(*                                                                            *)
(*  If the number of matches meets the minimum criterion as specified by the  *)
(*  user, then a record is dynamically created, storing the two item codes    *)
(*  which matched.                                                            *)
(*                                                                            *)
(*  1. Since dynamic variables are used, stack overflow may occur in certain  *)
(*     cases.  The program does not yet handle this.  When stack overflow     *)
(*     does occur, it usually means the minimum criterion is not rigid        *)
(*     enough, meaning the program is finding a lot of matches.               *)
(*     One solution is to raise the minimum criterion so more keywords must   *)
(*     match before a question is considered to be a match.                   *)
(*                                                                            *)
(*  2. This will not scan graphics questions.                                 *)
(*                                                                            *)
(******************************************************************************)


SEGMENT PROCEDURE SEARCHTEXT;

CONST (* maximum number of keys you can extract *)
      MAXKEYS = 10;

      (* question textfile control codes *)
      GOTOFLAG = 128;       (* flags a gotoxy *)
      PAGEFLAG = 129;       (* flags text continues on another page *)
      UNUSEDFLAG = 130;     (* flags unused byte *)
      ENDITEM = 131;        (* flags end of text for a question *)

      VERSION = 'Cat System Duplicate Question Search [1.03]';


TYPE (* key word *)
     KEYTYPE = PACKED ARRAY[0..7] OF CHAR;

     (* record pointer *)
     PPTR = ^PAIRS;

     (* information saved if question match occured *)
     PAIRS = RECORD
               ITEM1,             (* item code of question *)
               ITEM2 : INTEGER;
               NEXT : PPTR;       (* next in list of question pairs *)
             END;


VAR (* string character buffer *)

    C,                (* counts how many questions looked at *)
    NUMMATCHING,      (* number of keywords matching/question *)
    MINCRIT,          (* minumum # of keywords which must match *)
    MAXNUMKEYS,        (* maximum # of keywords to fetch/question *)
    SLOTNUM,           (* directory index *)
    BLK,               (* block where question text starts *)
    BLKPTR,            (* byte in block where question text starts *)
    I,                 (* misc counters *)
    K,
    L,
    M,
    N,
    DATASLOT : INTEGER;  (* record # where question data exists *)

    SCAN,                   (* true ==> scan the text/save keywords *)
    OFFSET,                 (* true ==> shift text over 40 columns *)
    CONSOLE : BOOLEAN;      (* true ==> send output to screen *)

    SELECT,
    LCOMMAND,
    OUTPUT : CHAR;

    (* key words for current question *)
    KEYWORD : ARRAY[1..MAXKEYS] OF RECORD
                                STR : KEYTYPE;
                                LENGTH : INTEGER;
                              END;


    (* list of keywords for subtest questions *)
    LIST : ARRAY[0..MAXITEMPOOL] OF RECORD
                                        WORD : PACKED ARRAY[0..79] OF CHAR;
                                      END;

    (* does this question have keywords ? Is it normal text, not grafix *)
    USED : PACKED ARRAY[0..MAXITEMPOOL] OF BOOLEAN;


    KEYSTR : KEYTYPE;

    (* pointers to list of matching questions *)
    (* p[i] points to head of the list with i key words matching *)
    P : ARRAY[1..MAXKEYS] OF RECORD
                              LPOINTER : PPTR;
                              LCOUNT : INTEGER;
                            END;

    WORDPAIR : PPTR;

    (* string variables which allow use of POS for pattern matching *)
    TEXTLINE : STRING[80];
    TKEY : STRING[8];
    TEXTKEYS : ARRAY[1..MAXKEYS] OF STRING[8];

    (* file of ascii codes, control #'s *)
    ITEMTEXT : FILE;

{.......................................................................}


    (* reads the item text file & displays item text .  This procedure      *)
    (* doubles as a scanner to extract key words and a procedure to display *)
    (* the text of a question. If SCAN is set to true, the key words will   *)
    (* be saved, else it just displays text on the screen.                  *)
    PROCEDURE SCANQUEST(BLOCKNUM,BLOCKPTR : INTEGER);
    CONST MAXBUFSIZE = 2047;
      BLOCKSOUT = 4;
    VAR STARTWORD,
        MIN,
        MINLOC,
        SLENGTH,
        X,
        Y,
        B,
        A,
        CURRPTR,
        CURRBLK,
        CHARCODE,
        CHARCNT : INTEGER;

        ENDLINE,
        ENDWORD,
        NOTBLANK,
        BADIO : BOOLEAN;

        (* buffer block of ascii, control codes, used to load question text  *)
        BUF : PACKED ARRAY[0..MAXBUFSIZE] OF 0..139;

        SHORTLINE : PACKED ARRAY[0..38] OF CHAR;

   {............................................................................}

        (* reads a block from disk into the item ascii buffer *)
        PROCEDURE READBLOCK(WHICHBLOCK : INTEGER);
        VAR BLOCKSTRANSFERRED,
            ERRNUM : INTEGER;

            BADIO : BOOLEAN;
        BEGIN
          BADIO := FALSE;
          RESET(ITEMTEXT,TEXTNAME);  (* question text *)
          BLOCKSTRANSFERRED := BLOCKREAD(ITEMTEXT,BUF,BLOCKSOUT,WHICHBLOCK);
          BADIO := ((BLOCKSTRANSFERRED < 1) OR (IORESULT <> 0));
          ERRNUM := IORESULT;
          CLOSE(ITEMTEXT,LOCK);
          IF BADIO THEN
            BEGIN {1}
              WRITELN;WRITELN;
              WRITE('Block read io error # ',ERRNUM);
              STALL;
              EXIT(PROGRAM);
            END;  {1}
        END;  (* read block *)

    {---------------------------------------------------------------------------}

        (* return the next code in ascii file *)
        FUNCTION BUFCODE : INTEGER;
        BEGIN
          BUFCODE := BUF[CURRPTR];
          CURRPTR := CURRPTR + 1;
          IF CURRPTR > MAXBUFSIZE THEN
          (* end of block/get next block and reset byte ptr *)
            BEGIN {1}
              CURRBLK := CURRBLK + BLOCKSOUT;
              READBLOCK(CURRBLK);
              CURRPTR := 0;
            END;  {1}
        END;  (* bufcode *)

{...............................................................................}

    BEGIN  (* scanquest *)


      IF SCAN THEN
        (* clear key word buffer for this question *)
        FOR I := 1 TO MAXKEYS DO
        BEGIN
          KEYWORD[I].STR := '        ';
          KEYWORD[I].LENGTH := 0;
        END;


      READBLOCK(BLOCKNUM);
      (* set block/byte ptrs *)
      CURRPTR := BLOCKPTR;
      CURRBLK := BLOCKNUM;
      FILLCHAR(LINEBUF[0],80,' ');
```

```
(* read bytes from the buffer *)
REPEAT

  (* get char from block buffer *)
  CHARCODE := bufcode;

  CASE CHARCODE OF
    GOTOFLAG : BEGIN {1}    (* move cursor *)
                 (* next two bytes after flag are x,y coord *)
                 X := BUFCODE;
                 Y := BUFCODE;
                 CHARCNT := BUFCODE;
                 IF (CURRPTR + CHARCNT - 1) > MAXBUFSIZE THEN
                 BEGIN {2}
                   B := (MAXBUFSIZE + 1) - CURRPTR;
                   MOVELEFT(BUF[CURRPTR],LINEBUF[X],B);
                   X := X + B;
                   B := CHARCNT - B;
                   CURRBLK := CURRBLK + BLOCKSOUT;
                   READBLOCK(CURRBLK);
                   CURRPTR := 0;
                   MOVELEFT(BUF[CURRPTR],LINEBUF[X],B);
                   CURRPTR := CURRPTR + B;
                 END   {2}
                 ELSE
                 BEGIN {3}
                   MOVELEFT(BUF[CURRPTR],LINEBUF[X],CHARCNT);
                   CURRPTR := CURRPTR + CHARCNT;
                   IF CURRPTR > MAXBUFSIZE THEN
                     BEGIN {4}
                       CURRBLK :=  CURRBLK + BLOCKSOUT;
                       CURRPTR := 0;
                       READBLOCK(CURRBLK);
                     END;  {4}
                 END;  {3}

                 IF NOT SCAN THEN
                 BEGIN
                   IF OFFSET THEN
                     GOTOXY(40,Y)
                   ELSE
                     GOTOXY(0,Y);

                   MOVELEFT(LINEBUF[0],SHORTLINE[0],39);

                   WRITE(SHORTLINE);
                   FILLCHAR(LINEBUF[0],80,' ');

                 END
                 ELSE
                 BEGIN
                   K := 0;
                   ENDLINE := FALSE;

                   REPEAT

                     (* skip past leading blanks to find a word *)
                     NOTBLANK := FALSE;
                     REPEAT
                       IF LINEBUF[K] = ' ' THEN
                         K := K + 1
                       ELSE
                         NOTBLANK := TRUE;
                     UNTIL (K > 79) OR (NOTBLANK);

                     IF K > 79 THEN
                       ENDLINE := TRUE
                     ELSE
                     BEGIN

                       (* extract word *)
                       SLENGTH := 0;
                       ENDWORD := FALSE;
```

```
                        STARTWORD := K;
                        REPEAT
                          SLENGTH := SLENGTH + 1;
                          K := K + 1;
                          IF K > 79 THEN
                            ENDLINE := TRUE
                          ELSE
                            IF LINEBUF[K] = ' ' THEN
                              ENDWORD := TRUE;
                        UNTIL (ENDLINE) OR (ENDWORD);
                        KEYSTR := '        ';
                        IF SLENGTH > 8 THEN
                          MOVELEFT(LINEBUF[STARTWORD],KEYSTR[0],8)
                        ELSE
                          MOVELEFT(LINEBUF[STARTWORD],KEYSTR[0],SLENGTH);

                       (* see if there are shorter words, if so replace *)
                        MIN := SLENGTH;
                        MINLOC := 0;
                        A := 1;
                        REPEAT
                          IF KEYWORD[A].LENGTH < MIN THEN
                          BEGIN
                            MIN := KEYWORD[A].LENGTH;
                            MINLOC := A;
                          END;
                          A := A + 1;
                        UNTIL (A > MAXNUMKEYS) OR (MIN = 0);

                        IF MINLOC <> 0 THEN
                        BEGIN
                          WRITE('*');

                          (* capitalize the word *)
                          FOR I := 0 TO 7 DO
                            IF KEYSTR[I] IN ['a'..'z'] THEN
                              KEYSTR[I] := CHR((ORD(KEYSTR[I]) - 32));
                          KEYWORD[MINLOC].STR := KEYSTR;
                          KEYWORD[MINLOC].LENGTH := SLENGTH;
                        END;
                      END;
                    UNTIL ENDLINE;
                    FILLCHAR(LINEBUF[0],80,' ');
                  END; (* if scan *)
                END;  (1)
      PAGEFLAG : ;
      ENDITEM  : ;

    END;
  UNTIL CHARCODE = ENDITEM; (* until end flag hit *)
  FILLCHAR(LINEBUF[0],80,' ');
END;  (* scanquest *)


(* compare the current question with others in list *)
PROCEDURE COMPARE;
VAR PLOC : INTEGER;
BEGIN

  (* start with first question to compare *)
  L := MAXSAMPLES + 1;
  WHILE (L < SLOTNUM) DO
  BEGIN

    (* if this question has keywords *)
    IF USED[L] THEN
    BEGIN
      WRITE('.');

      (* move the key words from master list into string var for POS *)
      MOVELEFT(LIST[L].WORD[0],TEXTLINE[1],80);
```

```
          (* compare current keys to keys loaded from list *)
          NUMMATCHING := 0;


          FOR N := 1 TO MAXNUMKEYS DO
          BEGIN
            PLOC := POS(TEXTKEYS[N],TEXTLINE);
            IF (PLOC <> 0) AND (TEXTKEYS[N] <> '        ')  THEN
            BEGIN
              NUMMATCHING := NUMMATCHING + 1;
              FILLCHAR(TEXTLINE[PLOC],8,' ');
            END;
          END;


          (* if number of keys meets minimum, save data *)
          IF NUMMATCHING >= MINCRIT THEN
          BEGIN
            NEW(WORDPAIR);
            WORDPAIR^.ITEM1 := SLOTNUM;
            WORDPAIR^.ITEM2 := L;
            WORDPAIR^.NEXT := P[NUMMATCHING].LPOINTER;
            P[NUMMATCHING].LPOINTER := WORDPAIR;
            P[NUMMATCHING].LCOUNT := P[NUMMATCHING].LCOUNT + 1;
            WRITE('.',NUMMATCHING,'.');
          END;

        END;

        L := L + 1;

      END;

    END;  (* compare *)



{------------------------------------------------------------------------}

    (* this procedure gets the search parameturs as desired *)
    PROCEDURE GETSEARCHINFO;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(20,0);
      WRITE('LEVEL OF ANALYSIS');
      GOTOXY(0,4);
      WRITELN(
'The level of analysis for question comparison is determined by the number');
      WRITELN(
'of key words fetched from each question.  If an analysis of 5 is used,');
       WRITELN(
'then the 5 longest words from each question will be used in comparing with');
       WRITELN(
'the 5 longest words from other questions.  You can have a MAXIMUM of 10');
       WRITELN('keys.');
       WRITELN;
       REPEAT
         WRITELN;
         WRITE('Enter number of keys to use and then press <RET> : ');
         READLN(MAXNUMKEYS);
         IF  (MAXNUMKEYS > MAXKEYS) OR (MAXNUMKEYS <= 0) THEN SQUAWK;
       UNTIL (MAXNUMKEYS <= MAXKEYS) AND (MAXNUMKEYS > 0);

      PAGE(OUTPUT);
      GOTOXY(20,0);
      WRITE('MINIMUM CRITERION');
      GOTOXY(0,4);
      WRITELN(
'The minimum criterion of analysis is the minimun number of keywords that ');
      WRITELN(
'match in order to consider the question as a possible duplicate.  If a');
      WRITELN(
'minimum criterion of 2 is used, then questions with a number of matching  ');
   WRITELN(
'keywords less than 2 are ignored.  Your minimum criterion should be less');
   WRITELN('than or equal to ',MAXNUMKEYS);
   WRITELN;
   REPEAT
    WRITELN;
    WRITELN('Enter minimum criterion of keys to consider,');
    WRITE('then press <RET> : ');
    READLN(MINCRIT);
    IF  (MINCRIT > MAXNUMKEYS) OR (MINCRIT <= 0) THEN SQUAWK;
   UNTIL (MINCRIT <= MAXNUMKEYS) AND (MINCRIT > 0);
  END;   (* get searchinfo *)


{-----------------------------------------------------------------------------}


  (* display data on matching questions *)
  PROCEDURE SHOWRESULTS;
  VAR PCOMMAND,
    LCOMMAND : CHAR;
    DSLOT,
    SNUM,
    INDEX : INTEGER;
    EXITLOOP,
    NOMATCH : BOOLEAN;

    {-------------------------------------------------------------------------}

    (* reads item text file & displays item text to printer or file *)
    PROCEDURE LISTPRINT(BLOCKNUM, BLOCKPTR : INTEGER);
    VAR X,
      Y,
      CURRPTR,
      CURRBLK,
      CHARCODE,
      SKIPBYTE : INTEGER;

      ITEMBUF : PACKED ARRAY[0..511] OF 0..139;


      (* reads a block from disk into the item ascii buffer *)
      PROCEDURE READITEMBLOCK(WHICHBLOCK : INTEGER);
      VAR BLOCKSTRANSFERRED,
        ERRNUM : INTEGER;

        BADIO : BOOLEAN;
      BEGIN
       BADIO := FALSE;
       RESET(ITEMTEXT,TEXTNAME);  (* question text *)
       BLOCKSTRANSFERRED := BLOCKREAD(ITEMTEXT,ITEMBUF,1,WHICHBLOCK);
       BADIO := ((BLOCKSTRANSFERRED < 1) OR (IORESULT <> 0));
       ERRNUM := IORESULT;
       CLOSE(ITEMTEXT,LOCK);
       IF BADIO THEN
        BEGIN {1}
         WRITELN;WRITELN;
         WRITE('Block read io error # ',ERRNUM);
         STALL;
         EXIT(PROGRAM);
        END;  {1}
      END;  (* readitemblock *)

    {.........................................................................}

      (* returns next code in file *)
      FUNCTION LBUFCODE : INTEGER;
      BEGIN
       LBUFCODE := ITEMBUF[CURRPTR];
```

```
          CURRPTR := CURRPTR + 1;
          IF CURRPTR > 511 THEN
          BEGIN {1}
            CURRBLK := CURRBLK + 1;
            READITEMBLOCK(CURRBLK);
            CURRPTR := 0;
          END;  {1}
        END;  (* lbufcode *)

    {.....................................................................}

    BEGIN  (*  listprint *)
      READITEMBLOCK(BLOCKNUM);
      CURRPTR := BLOCKPTR;
      CURRBLK := BLOCKNUM;
      REPEAT
        CHARCODE := LBUFCODE;
        CASE CHARCODE OF
          GOTOFLAG : BEGIN {1}
                       X := LBUFCODE;
                       Y := LBUFCODE;
                       (* ignore next byte, due to a file modification *)
                       SKIPBYTE := LBUFCODE;
                       WRITELN(DEST);
                       WRITE(DEST,' ' : X);
                     END;  {1}
          PAGEFLAG : BEGIN {3}
                       WRITELN(DEST);
                       WRITELN(DEST);
                       WRITELN(DEST);
                     END;  {3}
          ENDITEM  : ;
        END; (* cases *)
        IF (CHARCODE >= 32) AND (CHARCODE <= 126) THEN
        BEGIN {4}
          X := X + 1;
          WRITE(DEST,CHR(CHARCODE));
        END;  {4}
      UNTIL CHARCODE = ENDITEM;
    END;  (* print *)

{-----------------------------------------------------------------------}

    (* lists things to the console *)
    PROCEDURE LCONSOLE;
    VAR FIXCHAR : CHAR;
    BEGIN
      SCANQUEST(BLK,BLKPTR);
      IF OFFSET THEN
        GOTOXY(40,20)
      ELSE
        GOTOXY(0,20);
      WRITE('Item code : ',DIRECTORY.ITEMCODE[SNUM]);

      IF OFFSET THEN
        GOTOXY(40,21)
      ELSE
        GOTOXY(0,21);
      WRITE('Answer :');
      CASE ITEMINFO.ATYPE OF
        CHARVALUE : WRITELN(' ',ITEMINFO.ANSWER);
        INTVALUE  : WRITELN(' ',ITEMINFO.INTANSWER);
        SEVENCHR  : BEGIN
                      FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                        WRITE(' ',ITEMINFO.CHRANSWER[I]);
                      WRITELN;
                    END;
      END;  (* cases *)
    END; (* lconsole *)

{-----------------------------------------------------------------------}

    (* lists item text and data to file/printer *)
```

```
PROCEDURE LFILE;
BEGIN
  WRITELN(DEST);
  WRITELN(DEST,' Item code : ',DIRECTORY.ITEMCODE[SNUM]);
  WRITELN(DEST);
  LISTPRINT(BLK,BLKPTR);
  WRITELN(DEST);
  WRITELN(DEST);
  WRITE(DEST,' Answer(s) : ');
  CASE ITEMINFO.ATYPE OF
    CHARVALUE : WRITELN(DEST,ITEMINFO.ANSWER);
    INTVALUE  : WRITELN(DEST,ITEMINFO.INTANSWER);
    SEVENCHR  : BEGIN {4}
                  FOR I := 1 TO ITEMINFO.ANSWERCOUNT DO
                    WRITE(DEST,ITEMINFO.CHRANSWER[I],' ');
                  WRITELN(DEST);
                END;  {4}
  END;  (* cases *)
  WRITELN(DEST);
END;  (* lfile *)


(* -------------------------------------------------------------------*)

(* write question text *)
PROCEDURE TEXTOUTPUT;
BEGIN
  IF NOT CONSOLE THEN
  BEGIN
    WRITELN(DEST);
    WRITELN(DEST);
    FOR I := 1 TO 79 DO
      WRITE(DEST,'*');
    WRITELN(DEST);
    WRITELN(DEST);
  END;
  WORDPAIR := P[INDEX].LPOINTER;
  EXITLOOP := FALSE;
  WHILE (WORDPAIR <> NIL) AND (NOT EXITLOOP) DO
  BEGIN

    SNUM := WORDPAIR^.ITEM1;
    DSLOT := HASH(SNUM);
    SEEK(FILEITEMINFO,DSLOT);
    GET(FILEITEMINFO);
    ITEMINFO := FILEITEMINFO^;
    BLK := ITEMINFO.ITEMBLOCK;
    BLKPTR := ITEMINFO.ITEMPTR;
    OFFSET := FALSE;

    IF CONSOLE THEN
    BEGIN
      PAGE(OUTPUT);
      LCONSOLE
    END
    ELSE
    BEGIN
      LFILE;
      WRITELN(DEST);
      WRITELN(DEST);
      FOR I := 1 TO 39 DO
        WRITE(DEST,'- ');
      WRITELN(DEST);
      WRITELN(DEST);
    END;

    SNUM := WORDPAIR^.ITEM2;
    DSLOT := HASH(SNUM);
    SEEK(FILEITEMINFO,DSLOT);
    GET(FILEITEMINFO);
    ITEMINFO := FILEITEMINFO^;
    BLK := ITEMINFO.ITEMBLOCK;
    BLKPTR := ITEMINFO.ITEMPTR;
```

```
        OFFSET := TRUE;

        IF CONSOLE THEN
        BEGIN
          LCONSOLE;
          GOTOXY(0,23);
          WRITE('Press <RET> to continue, <ESC> to quit ');
          IF GETCHAR([CHR(RET),CHR(ESC)],TRUE,TRUE,TRUE) = CHR(ESC) THEN
            EXITLOOP := TRUE;
        END
        ELSE
        BEGIN
          LFILE;
          WRITELN(DEST);
          WRITELN(DEST);
          FOR I := 1 TO 79 DO
            WRITE(DEST,'*');
          WRITELN(DEST);
          WRITELN(DEST);
        END;

        WORDPAIR := WORDPAIR^.NEXT;
      END;
    END;  (* textoutput *)

(* ------------------------------------------------------------------- *)

    (* get the output destination for results *)
    PROCEDURE GETOUTINFO;
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(20,0);
      WRITE('OUTPUT SELECT MENU');
      GOTOXY(0,4);
      WRITE('Select one of the following options by entering its number.');
      GOTOXY(16,8);
      WRITE('1.  QUIT');
      GOTOXY(16,9);
      WRITE('2.  SEARCH RESULTS TO CONSOLE');
      GOTOXY(16,10);
      WRITE('3.  SEARCH RESULTS TO PRINTER');
      GOTOXY(16,11);
      WRITE('4.  SEARCH RESULTS TO FILE');
      GOTOXY(16,18);
      WRITE('Enter Choice # : ');
      SELECT :=  GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
      CONSOLE := FALSE;

      CASE SELECT OF
        '1' : ;
        '2' : BEGIN
                REWRITE(DEST,'CONSOLE:');
                CONSOLE := TRUE;
              END;
        '3' : REWRITE(DEST,UNITNUMPRINTER);
        '4' : GETNEWFILE;
      END;
    END;  (* getoutinfo *)


(* begin search results *)
BEGIN

  (* turn off scan flag so scan routine will display text instead of scan *)
  SCAN := FALSE;

  WRITE(CHR(1));

  NOMATCH := TRUE;

  REPEAT
```

```
 FOR INDEX := MAXNUMKEYS DOWNTO 1 DO
 BEGIN
   IF P[INDEX].LCOUNT > 0 THEN
   BEGIN
     NOMATCH := FALSE;
     PAGE(OUTPUT);
     WRITELN(P[INDEX].LCOUNT,' matches occured on ',INDEX,' keywords.');
     WRITELN;
     WRITE('Do you want to see the itemcodes of matches ?   Y/N  : ');
     LCOMMAND := GETCHAR(['Y','N','y','n'],TRUE,TRUE,TRUE);

     IF LCOMMAND IN ['y','Y'] THEN
     BEGIN
       WRITELN;
       WRITELN;
       WRITELN;
       WRITE('Would you like to see the question text also ?  Y/N  : ');
       PCOMMAND := GETCHAR(['Y','N','y','n'],TRUE,TRUE,TRUE);

       GETOUTINFO;

       IF SELECT <> '1' THEN
       BEGIN

         PAGE(OUTPUT);

         FOR K := 1 TO 79 DO
           WRITE(DEST,'*');
         WRITELN(DEST);
         WRITELN(DEST,DIRECTORY.TESTNAME);
         FOR K := 1 TO 79 DO
           WRITE(DEST,'*');
         WRITELN(DEST);
         WRITELN(DEST,'Level of Analysis : ',MAXNUMKEYS);
         WRITELN(DEST,'Minimum Criterion : ',MINCRIT);
       WRITELN(DEST,'Number of Matching Questions on ',INDEX,' Keys : ',
             P[INDEX].LCOUNT);
        WRITELN(DEST);

        WORDPAIR := P[INDEX].LPOINTER;
        L := 0;
        WHILE WORDPAIR <> NIL DO
        BEGIN
          L := L + 1;
          WRITE(DEST,DIRECTORY.ITEMCODE[WORDPAIR^.ITEM1],' / ',
                DIRECTORY.ITEMCODE[WORDPAIR^.ITEM2],'      ');
          WORDPAIR := WORDPAIR^.NEXT;
          IF L MOD 5 = 0 THEN WRITELN(DEST);
        END;

        WRITELN(DEST);
        WRITELN(DEST);

        IF CONSOLE THEN
          STALL;


        IF PCOMMAND IN ['Y','y'] THEN
          TEXTOUTPUT;

        CLOSE(DEST,LOCK);
      END;
    END;
  END;
END;

PAGE(OUTPUT);
IF NOMATCH THEN
BEGIN
  WRITELN('No matches occurred for this scan of ',DIRECTORY.TESTNAME);
  WRITELN;
  STALL;
```

```
         EXIT(SHOWRESULTS);
       END
       ELSE
         WRITE('Would you like to look at the results again ?     Y/N  : ');
     UNTIL  (GETCHAR(['Y','y','N','n'],TRUE,TRUE,TRUE) IN ['N','n']);
   END;  (* showresults *)

{------------------------------------------------------------------------------}


   (* initialize arrays *)
   PROCEDURE INITIALIZE;
   BEGIN
      FOR I := 1 TO MAXKEYS DO
      BEGIN
        P[I].LPOINTER := NIL;
        P[I].LCOUNT := 0;
      END;


      FOR I := 0 TO MAXITEMPOOL DO
        USED[I] := FALSE;

      TKEY := '          ';
      TEXTLINE :=
'                                                                                 ';

      PAGE(OUTPUT);
      GOTOXY(0,6);
      WRITE(VERSION);
      GOTOXY(0,8);
      WRITELN(DIRECTORY.TESTNAME);
      WRITELN('Level of Analysis : ',MAXNUMKEYS);
      WRITELN('Minimum Criterion : ',MINCRIT);
      GOTOXY(0,12);
      WRITE(CHR(2));
   END;     (* initialize *)


(*----------------------------------------------------------------*)

BEGIN  (* searchtext  *)

  (* set terminate program flag to false *)
  ESCPROC := FALSE;

  (* get the subtest you want to search *)
  LOADTEST('Search text in which subtest? : ');

  (* if you are not done *)
  IF NOT ESCPROC THEN
  BEGIN

    (* get # of keywords to fetch, criterion *)
    GETSEARCHINFO;

    (* set up data structures *)
    INITIALIZE;

    (* start with first question *)
    SLOTNUM := MAXSAMPLES + 1;

    (* open the question data file *)
    RESET(FILEITEMINFO,DATANAME);

    (* initialize count of number of questions processed *)
    C := 0;

    (* set flag for extracting key words *)
    SCAN := TRUE;

    (* look at all the questions in the subtest directory *)
    REPEAT
```

```
 (* if the item code is > 0 then question exists *)
 IF DIRECTORY.ITEMCODE[SLOTNUM] >= 0 THEN
 BEGIN {10}

   (* get data of question *)
   DATASLOT := HASH(SLOTNUM);
   SEEK(FILEITEMINFO,DATASLOT);
   GET(FILEITEMINFO);
   ITEMINFO := FILEITEMINFO^;


   (* if not a graphics question, then we can extract text *)
   IF NOT ITEMINFO.GRAPHICS THEN
   BEGIN

     (* get the text pointers *)
     BLK := ITEMINFO.ITEMBLOCK;
     BLKPTR := ITEMINFO.ITEMPTR;

     (* mark this question has keywords *)
     USED[SLOTNUM] := TRUE;

     (* count how many questions processed so far *)
     C := C + 1;

     (* indicate number of questions processed and memory available *)
     WRITELN;
     WRITE(C:4,'.    [',MEMAVAIL,'] ');


      (* look at the question and extract keywords *)
      SCANQUEST(BLK,BLKPTR);


      (* save the key words for this question in master list *)
      N := 0;
      FOR M := 1 TO MAXNUMKEYS DO
      BEGIN
        KEYSTR := KEYWORD[M].STR;
        MOVELEFT(KEYSTR[0],LINEBUF[N],8);
        N := N + 8;
      END;
      LIST[SLOTNUM].WORD := LINEBUF;

      (* move keyword in a list of "strings" so we can use the *)
      (* built in POS function to find keyword match           *)
      FOR N := 1 TO MAXNUMKEYS DO
      BEGIN
        KEYSTR := KEYWORD[N].STR;
        MOVELEFT(KEYSTR[0],TKEY[1],8);
        TEXTKEYS[N] := TKEY;
      END;

      (* compare our current keywords with master list *)
      (* if question match occurred, allocate new      *)
      (* record and store in appropriate list.         *)
      COMPARE;

    END;
  END;  {10}
  SLOTNUM := SLOTNUM + 1;
UNTIL (SLOTNUM > MAXITEMPOOL);

(* notify done with scan *)
FOR I := 1 TO 8 DO
  SQUAWK;

(* display the results of scan *)
SHOWRESULTS;

CLOSE(FILEITEMINFO,NORMAL);
```

```
   END;
END;    (* searchtext *)
```

```
(*****************************************************************)
(* FILE : D.INFOTAB.TEXT              "include file for D.MGR    *)
(* File last modified : Mar 11,1983                              *)
(*****************************************************************)

(* It is a verification that all items *)
(* in the info table are also in the subtest directories.                      *)
(* It works by loading an infotable then checking the subtest directory to     *)
(* see if the infotable entry exists in the subtest database.                  *)
SEGMENT PROCEDURE CHECKINFOTABS;

TYPE (* save locations in infotable where errors occured *)
     ERRMATRIX = PACKED ARRAY[1..INFOCOLUMN,1..INFOROW] OF BOOLEAN;

VAR LINE,      (* keep track of line number on screen *)
    ERRORCNT,  (* running count of errors *)
    DSLOT,     (* directory slot of item *)
    I,         (* various counters *)
    J,
    K : INTEGER;

    (* keeps tracks of infotable errors for each subtest *)
    ERRTAB : PACKED ARRAY[0..MAXSUBTESTS]  OF RECORD
                                             ECNT : INTEGER;
                                             ELIST : ERRMATRIX;
                                           END;

    (* list the errors in the infotable *)
    PROCEDURE LISTINFOERRS;
    VAR SELECT : CHAR;
        CONSOLE : BOOLEAN;
        HEADER : STRING;

        (* get the output destination *)
        PROCEDURE GETOUTDEST;
        BEGIN
          PAGE(OUTPUT);
          GOTOXY(20,0);
          WRITE('OUTPUT SELECT MENU');
          GOTOXY(0,4);
          WRITE('Select one of the following options by entering its number.');
          GOTOXY(16,8);
          WRITE('1.  QUIT');
          GOTOXY(16,9);
          WRITE('2.  INFOTABLE ERRORS TO CONSOLE');
          GOTOXY(16,10);
          WRITE('3.  INFOTABLE ERRORS TO PRINTER');
          GOTOXY(16,11);
          WRITE('4.  INFOTABLE ERRORS TO FILE');
          GOTOXY(16,18);
          WRITE('Enter Choice # : ');
          SELECT := GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
          CONSOLE := FALSE;

          CASE SELECT OF
            '1' : EXIT(CHECKINFOTABS);
            '2' : BEGIN
                    CONSOLE := TRUE;
                    REWRITE(DEST,'CONSOLE:');
                  END;
            '3' : REWRITE(DEST,UNITNUMPRINTER);
            '4' : GETNEWFILE;
          END;
        END; (* getoutdest *)

    BEGIN  (* listinfoerrs *)

      (* get information to send output destination *)
      GETOUTDEST;

      (* write a header *)
      PAGE(OUTPUT);
      HEADER :=
'ITEMCODE  ROW  COLUMN      ITEMCODE  ROW  COLUMN      ITEMCODE  ROW  COLUMN';

      (* open the subtest directories file *)
      RESET(FILEDIRECTORY,INDEXNAME);

      (* look at each subtest *)
      FOR CURRINDEXRECNUM := 0 TO 20 DO
      BEGIN

        (* if there were errors, display them *)
        IF ERRTAB[CURRINDEXRECNUM].ECNT > 0 THEN
        BEGIN

          (* get the subtest directory *)
          SEEK(FILEDIRECTORY,CURRINDEXRECNUM);
          GET(FILEDIRECTORY);

          (* if a subtests exists here *)
          IF NOT (FILEDIRECTORY^.UNUSED) THEN
          BEGIN

            (* get the subtest directory *)
            DIRECTORY := FILEDIRECTORY^;

            (* write some diagnostic info *)
            WRITELN(DEST);
            WRITELN(DEST);
            WRITELN(DEST);
            PAGE(OUTPUT);
            WRITELN(DEST,'Subtest : ',DIRECTORY.TESTNAME);
            WRITELN(DEST,'This items do not exist in database.');
            WRITELN(DEST);
            WRITELN(DEST,HEADER);
            LINE := 1;

            (* load the infotable for this subtest *)
            LOADINFO(CURRINDEXRECNUM);

            (* check the error matrix and display errors *)
            FOR J := 1 TO INFOROW DO
              FOR I := 1 TO INFOCOLUMN DO
              BEGIN
                IF (ERRTAB[CURRINDEXRECNUM].ELIST[I,J]) THEN
                BEGIN

                  (* make it look nice on the screen *)
                  IF (LINE MOD 3 = 0) THEN
                    WRITELN(DEST,INFOTABLE[I,J] : 6, I:7, J:6)
                  ELSE
                    WRITE(DEST,INFOTABLE[I,J] : 6, I:7, J:6,'        ');
                  LINE := LINE + 1;

                  (* let user control screen scrolling *)
                  IF (CONSOLE) AND (LINE = 55) THEN
                  BEGIN
                    WRITELN;
                    WRITE('Press <RET> to continue or <ESC> to quit : ');
                    IF GETCHAR([CHR(RET),CHR(ESC)],TRUE,TRUE,TRUE) = CHR(ESC)
                    THEN
                    BEGIN
                      CLOSE(DEST,NORMAL);
                      EXIT(CHECKINFOTABS);
                    END;

                    (* start new screen page, write header again *)
                    PAGE(OUTPUT);
                    WRITELN(DEST,'Subtest : ',DIRECTORY.TESTNAME);
                    WRITELN(DEST,'This items do not exist in database.');
                    WRITELN(DEST);
                    WRITELN(DEST,HEADER);
                    LINE := 1;
                  END;
                END;
              END;

              (* last screen page, wait till user wishes to continue *)
              IF (CONSOLE) AND (LINE <> 1) THEN
              BEGIN
                WRITELN;
                WRITE('Press <RET> to continue or <ESC> to quit : ');
                IF GETCHAR([CHR(RET),CHR(ESC)],TRUE,TRUE,TRUE) = CHR(ESC)
                THEN
                BEGIN
                  CLOSE(DEST,NORMAL);
                  EXIT(CHECKINFOTABS);
                END;
              END;
          END;
        END;
      END;
      CLOSE(DEST,NORMAL);
      CLOSE(FILEDIRECTORY,NORMAL);
    END;   (* list infoerrs  *)

BEGIN
  PAGE(OUTPUT);

  (* open the subtest directories file *)
  RESET(FILEDIRECTORY,INDEXNAME);

  (* write a header *)
  LINE := 2;
  WRITE('      SUBTEST                         # of infotable errors');

  (* initialize error counts for each subtest *)
  FOR I := 0 TO MAXSUBTESTS DO
    ERRTAB[I].ECNT := 0;
  GOTOXY(0,2);

  (* step through the subtest directories file *)
  FOR CURRINDEXRECNUM := 0 TO MAXSUBTESTS DO
  BEGIN
    SEEK(FILEDIRECTORY,CURRINDEXRECNUM);
    GET(FILEDIRECTORY);

    (* if a subtests exists here *)
    IF NOT (FILEDIRECTORY^.UNUSED) THEN
    BEGIN

      (* get the directory *)
      DIRECTORY := FILEDIRECTORY^;

      (* load infotable for this subtest *)
      LOADINFO(CURRINDEXRECNUM);

      ERRORCNT := 0;  (* initialize running count of errors *)

      (* look through the entire infotable *)
      FOR J := 1 TO INFOROW DO
      BEGIN
        WRITE('.');
        FOR I := 1 TO INFOCOLUMN DO
        BEGIN

          (* if infotable has no value *)
          IF INFOTABLE[I,J] < 0 THEN
            DSLOT := -1
          ELSE
            (* return the slot number in directory if any *)
            DSLOT := SLOT(INFOTABLE[I,J]);

          (* negative value means there was no such item in directory *)
          IF DSLOT < 0 THEN
          BEGIN
            ERRORCNT := ERRORCNT + 1;  (* increment error count *)
            (* update error matrix *)
            ERRTAB[CURRINDEXRECNUM].ELIST[I,J] := TRUE;
          END
          ELSE
            ERRTAB[CURRINDEXRECNUM].ELIST[I,J] := FALSE;
        END;
      END;

      (* update total errors for this subtest *)
      ERRTAB[CURRINDEXRECNUM].ECNT := ERRORCNT;
      BLANKLINES(LINE,1,LINE);
      WRITE(FILEDIRECTORY^.TESTNAME);
      GOTOXY(50,LINE);
      WRITELN(ERRORCNT);
      LINE := LINE + 1;
    END;
  END;
  CLOSE(FILEDIRECTORY,NORMAL);
  WRITE('Do you want to list the errors ?   Y/N : ');
  IF GETCHAR(['Y','N','y','n'],TRUE,TRUE,TRUE) IN ['Y','y'] THEN
    LISTINFOERRS;
END;  (* Checkinfotab   *)
```

```
(*****************************************************************)
(* FILE : D.GRAFIX.TEXT             *include file for D.MGR     *)
(* File last modified : Mar 11,1983                              *)
(*****************************************************************)

(*************************************************************)
(* checks to see that if the graphics flag is set, there is *)
(* a foto file or compressed file out on the volume catfoto  *)
(* It works by turning off the i/o checking , then attempting *)
(* to reset the grafix file in question.  If the i/o error was *)
(* zero, then the graphics file existed, if not, then we keep  *)
(* track of the errors.                                        *)
(***************************************************************)
SEGMENT PROCEDURE CHECKGFILES;

TYPE (* keeps track of errors for each item *)
     ERRLIST = PACKED ARRAY[0..MAXITEMPOOL] OF BOOLEAN;

VAR X,Z,           (* used as counters *)
    LINE,          (* keeps track of line on screen *)
    CERRORS,       (* count of compressed grafix errors *)
    FERRORS,       (* count of normal fotofile errors *)
    GCNT,          (* total grafix  count in subtest *)
    I,             (* various counters *)
    J : INTEGER;

    STR,           (* used to construct filename for grafixs *)
    DIGITSTR,      (* from the itemcode.                     *)
    FNAME : STRING;
    C : CHAR;

    (* keeps track of errors for each subtest *)
    GERRORS : PACKED ARRAY[0..MAXSUBTESTS] OF RECORD
                                              (* number of errors *)
                                              GERRCNT : INTEGER;
                                              (* graphic error *)
                                              FLIST,
                                              (* krunched error *)
                                              KLIST : ERRLIST;
                                            END;

    (* list the errors in the infotable *)
    PROCEDURE LISTFOTOERRS;
    VAR SELECT : CHAR;
        CONSOLE : BOOLEAN;
        HEADER : STRING;

        (* get the output destination *)
        PROCEDURE GETOUTDEST;
        BEGIN
          PAGE(OUTPUT);
          GOTOXY(20,0);
          WRITE('OUTPUT SELECT MENU');
          GOTOXY(0,4);
          WRITE('Select one of the following options by entering its number.');
          GOTOXY(16,8);
          WRITE('1.  QUIT');
          GOTOXY(16,9);
          WRITE('2.  GRAPHICS ERRORS TO CONSOLE');
          GOTOXY(16,10);
          WRITE('3.  GRAPHICS ERRORS TO PRINTER');
          GOTOXY(16,11);
          WRITE('4.  GRAPHICS ERRORS TO FILE');
          GOTOXY(16,18);
          WRITE('Enter Choice # : ');
          SELECT := GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
          CONSOLE := FALSE;

          CASE SELECT OF
            '1' : EXIT(CHECKGFILES);
            '2' : BEGIN
                    CONSOLE := TRUE;
                    REWRITE(DEST,'CONSOLE:');
```

```
                 END;
           '3' : REWRITE(DEST,UNITNUMPRINTER);
           '4' : GETNEWFILE;
         END;
       END;  (* getoutdest *)

   BEGIN  (* listfotoerrs *)

     (* get destination to send output *)
     GETOUTDEST;
     PAGE(OUTPUT);

     (* open the subtest directories file *)
     RESET(FILEDIRECTORY,INDEXNAME);

     (* scan each subtest *)
     FOR CURRINDEXRECNUM := 0 TO 20 DO
     BEGIN

       (* if there were some errors *)
       IF GERRORS[CURRINDEXRECNUM].GERRCNT > 0 THEN
       BEGIN

         (* get the subtest directory *)
         SEEK(FILEDIRECTORY,CURRINDEXRECNUM);
         GET(FILEDIRECTORY);
         IF NOT (FILEDIRECTORY^.UNUSED) THEN
         BEGIN
           DIRECTORY := FILEDIRECTORY^;

           (* write a header *)
           WRITELN(DEST);
           WRITELN(DEST);
           WRITELN(DEST);
           PAGE(OUTPUT);
           WRITELN(DEST,'Subtest : ',DIRECTORY.TESTNAME);
           WRITELN(DEST,
           'This items are missing foto files or compressed graphic files.');
           WRITELN(DEST);
           WRITELN(DEST,
'ITEMCODE     MISSING GRAPHICS              ITEMCODE     MISSING GRAPHICS');
           LINE := 1;

           (* scan the error list *)
           FOR J := 1 TO MAXITEMPOOL DO
           BEGIN

             (* if an item existed in this directory slot and there *)
             (* was a graphics error.                               *)
             IF (GERRORS[CURRINDEXRECNUM].FLIST[J]) AND
                (DIRECTORY.ITEMCODE[J] > 0) THEN
             BEGIN

                 (* if the error was a compressed grafix error *)
                 IF (GERRORS[CURRINDEXRECNUM].KLIST[J]) THEN
                   WRITE(DEST,DIRECTORY.ITEMCODE[J] :  6,
                   '          compressed file')
                 ELSE
                   (* the error was a fotofile error *)
                   WRITE(DEST,DIRECTORY.ITEMCODE[J] :  6,
                   '          fotofile      ');
                 LINE := LINE + 1;

                 (* format it so it looks nice *)
                 IF ODD(LINE) THEN
                   WRITELN(DEST)
                 ELSE
                   WRITE(DEST,'          ');
             END;

             (* allow user to see screen without scrolling *)
```

```
              IF (CONSOLE) AND (LINE = 38) THEN
              BEGIN
                WRITELN;
                WRITE('Press <RET> to continue or <ESC> to quit : ');
                IF GETCHAR([CHR(RET),CHR(ESC)],TRUE,TRUE,TRUE) = CHR(ESC)
                THEN
                BEGIN
                  CLOSE(DEST,NORMAL);
                  EXIT(CHECKGFILES);
                END;

                (* write a new page of output to screen and the header *)
                PAGE(OUTPUT);
                WRITELN('Subtest : ',DIRECTORY.TESTNAME);
                WRITELN(
          'This items are missing foto files or compressed graphic files.');
                WRITELN;
                WRITELN(
'ITEMCODE     MISSING GRAPHICS                  ITEMCODE     MISSING GRAPHICS');
                LINE := 1;
              END;
            END;

            (* allow user to see the last page without scrolling *)
            IF (CONSOLE) AND (LINE <> 1) THEN
            BEGIN
              WRITELN;
              WRITE('Press <RET> to continue or <ESC> to quit : ');
              IF GETCHAR([CHR(RET),CHR(ESC)],TRUE,TRUE,TRUE) = CHR(ESC)
              THEN
              BEGIN
                CLOSE(DEST,NORMAL);
                EXIT(CHECKGFILES);
              END;
            END;
          END;
        END;
      END;
      CLOSE(DEST,NORMAL);
      CLOSE(FILEDIRECTORY,NORMAL);
    END;   (* list FOTOerrs  *)


    (* searches through the directory looking for questions *)
    (* with graphics flags on, then checks if file is on volume *)
    PROCEDURE GSEARCH;
    VAR DATASLOT : INTEGER;
    BEGIN

      (* open the question data files *)
      RESET(FILEITEMINFO,DATANAME);

      (* scan the entire directory *)
      FOR J := 1 TO MAXITEMPOOL DO
      BEGIN

        (* initialize error flags *)
        GERRORS[CURRINDEXRECNUM].FLIST[J] := FALSE;
        GERRORS[CURRINDEXRECNUM].KLIST[J] := FALSE;

        (* indicate that something is happening *)
        IF J MOD 10 = 0 THEN
          WRITE('.');

        (* if an item exists in this data slot *)
        IF DIRECTORY.ITEMCODE[J] > 0 THEN
        BEGIN

          (* get the record number where its data exists *)
          DATASLOT := HASH(J);

          (* get its data *)
          SEEK(FILEITEMINFO,DATASLOT);
```

```
GET(FILEITEMINFO);
ITEMINFO := FILEITEMINFO^;

(* if the graphics flag is set *)
IF ITEMINFO.GRAPHICS THEN
BEGIN

  (* update count of graphics for this subtest *)
  GCNT := GCNT + 1;

  (* make the file name for the graphic question *)
  (* change an integer to a string value *)
  DIGITSTR := ' ';
  STR := '';
  IF J > 5 THEN
    Z := DIRECTORY.ITEMCODE[J]
  ELSE
    Z := J;

  REPEAT
    X := Z MOD 10;
    C := CHR(X+48);
    DIGITSTR[1] := C;
    STR := CONCAT(DIGITSTR,STR);
    Z := Z DIV 10;
  UNTIL Z <= 0;
  DIGITSTR[1] := CHR(CURRINDEXRECNUM+65);
  C := DIGITSTR[1];

  (* if the question is a compressed grafix *)
  IF ITEMINFO.DUMMY1 = 1.0 THEN
  BEGIN

    (* check if sample question *)
    IF J < 6 THEN (* it is sample question *)
      FNAME := CONCAT('/CATFOTO/',DIGITSTR,'DIR/G',
                       DIGITSTR,'SQ',STR,'.DATA')
    ELSE
      FNAME := CONCAT('/CATFOTO/',DIGITSTR,'DIR/G',
                       DIGITSTR,STR,'.DATA');

    (* see if it is out on catfoto volume *)
    (*$I-*)
      RESET(DEST,FNAME);
    (*$I+*)

      (* error occured *)
      IF IORESULT <> 0 THEN
      BEGIN
        CERRORS := CERRORS + 1;
        GERRORS[CURRINDEXRECNUM].FLIST[J] := TRUE;
        GERRORS[CURRINDEXRECNUM].KLIST[J] := TRUE;
      END;

      CLOSE(DEST,NORMAL);
  END
  ELSE
  BEGIN
    (* check if sample question *)
    IF J < 6 THEN (* it was sample question *)
      FNAME := CONCAT('/CATFOTO/',DIGITSTR,'DIR/G',
                       DIGITSTR,'SQ',STR,'.FOTO')
    ELSE
      FNAME := CONCAT('/CATFOTO/',DIGITSTR,'DIR/G',
                       DIGITSTR,'Q',STR,'.FOTO');

    (* see if it is out on disk *)
    (*$I-*)
      RESET(DEST,FNAME);
    (*$I+*)

      (* error occured *)
      IF IORESULT <> 0 THEN
```

```
                BEGIN
                  FERRORS := FERRORS + 1;
                  GERRORS[CURRINDEXRECNUM].FLIST[J] := TRUE;
                END;

                CLOSE(DEST,NORMAL);
            END;     (* if it was a normal or compressed graphics *)
          END;    (* if it was a graphics *)
        END;   (* if there was an item at this slot *)
      END;  (* loop which looks at entire directory *)

      CLOSE(FILEITEMINFO,NORMAL);
    END;  (* gsearch *)


BEGIN
  PAGE(OUTPUT);

  (* open the subtest directories file *)
  RESET(FILEDIRECTORY,INDEXNAME);

  (* write a diagnostic header *)
  LINE := 2;
  WRITE(
'      SUBTEST                      # of graphics     foto errs   krunch  errs');
  GOTOXY(0,2);


  (* look at each subtest directory record *)
  FOR CURRINDEXRECNUM := 0 TO MAXSUBTESTS DO
  BEGIN

    (* initialize error count *)
    GERRORS[CURRINDEXRECNUM].GERRCNT := 0;

    (* check if there is a subtest *)
    SEEK(FILEDIRECTORY,CURRINDEXRECNUM);
    GET(FILEDIRECTORY);
    IF NOT (FILEDIRECTORY^.UNUSED) THEN
    BEGIN

      (* get the subtest directory *)
      DIRECTORY := FILEDIRECTORY^;

      (* initialize errors *)
      FERRORS := 0;
      CERRORS := 0;
      GCNT := 0;

      (* look at each item, if the grafix flag is set, and if so *)
      (* if there is a graphics file for the question.           *)
      GSEARCH;

      (* write out diagnostic information *)
      BLANKLINES(LINE,1,LINE);
      WRITE(FILEDIRECTORY^.TESTNAME);
      GOTOXY(42,LINE);
      WRITE(GCNT);
      GOTOXY(55,LINE);
      WRITE(FERRORS);
      GOTOXY(70,LINE);
      WRITELN(CERRORS);
      LINE := LINE + 1;
      GERRORS[CURRINDEXRECNUM].GERRCNT := CERRORS + FERRORS;
    END;
  END;
  CLOSE(FILEDIRECTORY,NORMAL);
  CLOSE(OUTFILE,NORMAL);
  WRITE('Do you want to list the errors ?   Y/N : ');
  IF GETCHAR(['Y','N','y','n'],TRUE,TRUE,TRUE) IN ['Y','y'] THEN
    LISTFOTOERRS;
END;  (* CheckGFILES    *)
```

MISC.DIR:
Subdirectory - Miscellaneous Textfiles

```
(*$S+*)
(******************************************************************************)
(*                                                                            *)
(*      Textfile : MISC/CATFFORMAT.TEXT         Volume : TFILES               *)
(*      Codefile : KATFFORMAT.CODE              Volume : CATDATA              *)
(*                                                                            *)
(******************************************************************************)
(*              DEC.  1, 1982                   NPROC                         *)
(******************************************************************************)
(*                                                                            *)
(* Description :                                                              *)
(*      This program formats the files needed to start up a CAT system.  It   *)
(* initializes the files to a prescribed size .  The files it will set up     *)
(* are those files needed for the subtest database, the examinee database,    *)
(* the system startup datafile, and the infotable datafiles.  This program    *)
(* should be used with caution because initializing files means that ANY      *)
(* EXISTING FILES WHICH ARE REINITIALIZED WILL CAUSE EXISTING DATA TO BE LOST.*)
(*                                                                            *)
(******************************************************************************)

PROGRAM FORMATFILES;

CONST (* ascii values *)
      BELL         = 7;
      NIL          = -1;
      MAXITEMBUF   = 511;
      GMAXSUBTEST  = 20;

TYPE (* different types of ways to answer a question *)
     ITEMRESPONSES = (CHARVALUE,      (* normal multiple choice *)
                      INTVALUE,       (* integer value as answer *)
                      SEVENCHR);      (* seven characters saved as answer *)


     (* type of question response *)
     SEVENTYPE = PACKED ARRAY[1..7] OF CHAR;


     (* all characters *)
     SETOFCHAR = SET OF CHAR;


VAR COMMAND,
    OUTPUT : char;              (* dummy variable for page function *)


    (* read an acceptable character from the keyboard *)
    FUNCTION GETCHAR(OKSET : SETOFCHAR;
                     FLUSHQUEUE,ECHO,BEEP : BOOLEAN) : CHAR;
    VAR MASK : PACKED ARRAY[0..0] OF CHAR;
    BEGIN
      IF FLUSHQUEUE THEN UNITCLEAR(2); (* flush buffer *)
      REPEAT
        UNITREAD(2,MASK,1);
        IF BEEP AND NOT (MASK[0] IN OKSET) THEN WRITE(CHR(BELL));
      UNTIL MASK[0] IN OKSET;
      IF ECHO AND (MASK[0] IN [CHR(32)..CHR(126)]) THEN
        WRITE(MASK[0]);
      GETCHAR := MASK[0];
    END;  (* getchar *)


    (* clear screen *)
    PROCEDURE PAGE(DUMMY : CHAR);
    BEGIN
      WRITE(CHR(28));
      GOTOXY(0,0);
    END;   (* page *)


    (* initialize the test directory *)
```

```
PROCEDURE SUBTESTFORMAT;

CONST DATANAME  = 'CATDATA:ITEMPOOL.DATA';     (* question data *)
      TEXTNAME  = 'QTEXT:ITEMTEXT.DATA';       (* question text *)
      INDEXNAME = 'CATDATA:TESTINDEX.DATA';    (* test directory *)

       (* slots available in test directory,  0 - 20 *)
      MAXSUBTESTS = 20;

       (* maximum question pool per test, 0 - 300 *)
      MAXITEMPOOL = 300;

TYPE (* test directory *)
     DIRDATA = PACKED RECORD
                         UNUSED     : BOOLEAN;
                         TESTNAME   : STRING;
                         ITEMCODE   : PACKED ARRAY
                                       [0..MAXITEMPOOL]
                                       OF INTEGER;  (* question code # *)
                       END;

     (* question ptrs/data , information for each question *)
     ITEMDATA = PACKED RECORD
                          GRAPHICS : BOOLEAN;
                          LOWANSWER,
                          HIGHANSWER : CHAR;
                          ITEMBLOCK,
                          ITEMPTR,
                          ANSWERCOUNT : INTEGER;
                          A,B,C,
                          PROPCORRECT,
                          POINTBISERIAL,
                          YOPT,
                          XOPT,
                          DUMMY1,
                          DUMMY2,
                          DUMMY3 : REAL;
                          CASE ATYPE : ITEMRESPONSES OF
                            CHARVALUE : (ANSWER : CHAR);
                            INTVALUE  : (INTANSWER : INTEGER);
                            SEVENCHR  : (CHRANSWER : SEVENTYPE);
                        END;

VAR I : INTEGER;
    CURRBLOCK,
    CURRFREEPTR,
    CURRINDEXRECNUM : INTEGER;


    (* block of ascii, control codes *)
    ITEMBUF : PACKED ARRAY[0..MAXITEMBUF] OF 0..139;


    (* test directory *)
    DIRECTORY : DIRDATA;
    FILEDIRECTORY : FILE OF DIRDATA;


    (* test question ptrs/data *)
    ITEMINFO : ITEMDATA;
    FILEITEMINFO : FILE OF ITEMDATA;


    (* file of ascii codes, control #'s *)
    ITEMTEXT : FILE;


     (* with data for that question           *)
     FUNCTION HASH(KEY : INTEGER) : INTEGER;
     BEGIN
       HASH :=
         (CURRINDEXRECNUM * MAXITEMPOOL)
         + KEY + CURRINDEXRECNUM;
```

```
END;  (* hash *)

(* saves value of free space, block & byte ptrs)
(* in block 0, bytes 0..3 of text file         *)
PROCEDURE SAVEPTRS;
VAR TRIX : RECORD CASE INTEGER OF
             1 : (TWOBYTES : PACKED ARRAY
                                [0..1] OF CHAR);
             2 : (INTVALUE : INTEGER);
           END;


    (* writes the item ascii buffer to diskfile  *)
    PROCEDURE WRITEITEMBLOCK(WHICHBLOCK : INTEGER);
    VAR BLOCKSTRANSFERRED,
        ERRNUM : INTEGER;
        BADIO : BOOLEAN;
    BEGIN
      BADIO := FALSE;
      RESET(ITEMTEXT,TEXTNAME);
      BLOCKSTRANSFERRED :=
        BLOCKWRITE(ITEMTEXT,ITEMBUF,1,WHICHBLOCK);
      BADIO := ((BLOCKSTRANSFERRED < 1) OR (IORESULT <> 0));
      ERRNUM := IORESULT;
      CLOSE(ITEMTEXT,LOCK);
      IF BADIO THEN
      BEGIN
        WRITELN;WRITELN;
        WRITE('Block write io error # ',ERRNUM);
        WRITELN;
        WRITELN('Possibly no room to expand ',TEXTNAME);
        WRITELN('Must have unused space at end of file');
        WRITELN('or put file at end of directory.');
        WRITELN;
        READLN;
        EXIT(PROGRAM);
      END;
    END;  (* writeitemblock *)


    (* reads a block from disk into the item ascii buffer *)
    PROCEDURE READITEMBLOCK(WHICHBLOCK : INTEGER);
    VAR BLOCKSTRANSFERRED,
        ERRNUM : INTEGER;
        BADIO : BOOLEAN;
    BEGIN
      BADIO := FALSE;
      RESET(ITEMTEXT,TEXTNAME);
      BLOCKSTRANSFERRED :=
        BLOCKREAD(ITEMTEXT,ITEMBUF,1,WHICHBLOCK);
      BADIO := ((BLOCKSTRANSFERRED < 1) OR (IORESULT <> 0));
      ERRNUM := IORESULT;
      CLOSE(ITEMTEXT,LOCK);
      IF BADIO THEN
      BEGIN
        WRITELN;WRITELN;
        WRITE('Block read io error # ',ERRNUM);
        WRITELN;
        WRITELN('Cant read ',TEXTNAME);
        WRITELN;
        READLN;
        EXIT(PROGRAM);
      END;
    END;

BEGIN   (* saveptrs *)
  READITEMBLOCK(0);
  TRIX.INTVALUE := CURRBLOCK;
  MOVELEFT(TRIX.TWOBYTES[0],ITEMBUF[0],2);
  TRIX.INTVALUE := CURRFREEPTR;
  MOVELEFT(TRIX.TWOBYTES[0],ITEMBUF[2],2);
  WRITEITEMBLOCK(0);
```

```
    END;   (* save ptrs *)

BEGIN   (* stfileformat *)
  PAGE(OUTPUT);
  WRITELN;
  WRITE(' Are you sure you want');
  WRITE(' to zero the directory ?  Y/N ');
  IF GETCHAR(['y','n','Y','N'],TRUE,FALSE,TRUE) IN ['Y','y'] THEN
  BEGIN
    WRITELN;
    WRITELN;
    WRITE(' Zeroing directory')·
    DIRECTORY.UNUSED := TRUE;
    FOR I := 0 TO MAXITEMPOOL DO
      DIRECTORY.ITEMCODE[I] := NIL;
    REWRITE(FILEDIRECTORY,INDEXNAME);
    FOR I := 0 TO MAXSUBTESTS DO
    BEGIN
      WRITE('.');
      SEEK(FILEDIRECTORY,I);
      FILEDIRECTORY^ := DIRECTORY;
      PUT(FILEDIRECTORY);
    END;
    CLOSE(FILEDIRECTORY,LOCK)·
    CURRINDEXRECNUM := MAXSUBTESTS - 1;
    I := HASH(MAXITEMPOOL);
    REWRITE(FILEITEMINFO,DATANAME);
    SEEK(FILEITEMINFO,I);
    CLOSE(FILEITEMINFO,LOCK);
    REWRITE(ITEMTEXT,TEXTNAME);
    I := BLOCKWRITE(ITEMTEXT,ITEMBUF,1,0);
    CLOSE(ITEMTEXT,LOCK);
    CURRBLOCK := 0;
    CURRFREEPTR := 4;
    SAVEPTRS;
  END;
END; (* stzerodirectory *)


(**** initialize the examinee files ****)
PROCEDURE EFORMAT;

CONST EINDEX      = 'CATDATA:EINDEX.DATA';        (* examinee directory *)
      INFONAME    = 'CATDATA:EINFO.DATA';         (* examinee data *)
      RESULTS     = 'CATDATA:ERESULTS.DATA';      (* examinee test scores *)
      PINFONAME   = 'CATDATA:EPDATA.DATA';        (* examinee personal data *)
      DONEINDEX   = 'CATDATA:DONE-DIR.DATA';      (* done examinee directory *)
      DONEINFO    = 'CATDATA:DONE-INFO.DATA';     (* done examinee test info *)
      DONERESULTS = 'CATDATA:DONE-RSLTS.DATA';(* done  test results *)
      DONEPINFO   = 'CATDATA:DONE-EPD.DATA';      (* done  personal data *)

      (* slots available in examinee directory, 0 - 50 *)
      MAXEXAMINEE = 50;

      (* max # of questions you can give per test *)
      QUESTIONS   = 20;


TYPE (* social security number *)
      IDTYPE = PACKED ARRAY[0..8] OF CHAR;

     (* examinee directory *)
     INDEX = PACKED ARRAY[0..MAXEXAMINEE] OF PACKED RECORD
                                             UNUSED : BOOLEAN;
                                             ID : IDTYPE;
                                           END;


    (* examinee testing  data *)
    EXAMEINFO = PACKED RECORD
                 ID           : IDTYPE;
```

```
              ORIENTATION_TIME,
              PREV_TIME_LASTTEST,
              NUMPROC,
              TOTTIMECONSOLE,
              NUMERRORS,
              LASTTEST      : INTEGER;
              DATE          : PACKED ARRAY[0..5] OF CHAR;
              TIME          : PACKED ARRAY[0..3] OF CHAR;
              TESTORDER,
              STRATEGY,
              TESTLENGTH    : PACKED ARRAY[1..GMAXSUBTEST]
                              OF 0..128;
              CKERROR       : ARRAY[1..GMAXSUBTEST] OF REAL;
              SUBSTOP       : PACKED ARRAY[1..GMAXSUBTEST] OF BOOLEAN;
            END;

(* question scores , data on examinee with respect to question *)
ITEM = PACKED RECORD
         ACORRECT : PACKED ARRAY[0..6] OF BOOLEAN;
         ACOUNT,
         ITEMNUM  : INTEGER;
         CORRECT  : BOOLEAN;
         THETA,
         ERROR,
         LATENCY  : REAL;
         CASE RTYPE : ITEMRESPONSES OF
           CHARVALUE : (RESPONSE : CHAR);
           INTVALUE  : (INTRESPONSE : INTEGER);
           SEVENCHR  : (CHRRESPONSE : SEVENTYPE);
       END;

(* test scores of examinee results  *)
SUBTEST = PACKED RECORD
            STTIME,
            STINSTRTIME,
            STPROCTCALLS,
            STERRKEY       : INTEGER;
            NUMITEMS,
            NUMCORR : 0..128;
            ESTABILITY,
            VARIANCE    : REAL;
            ITEMINFO    : PACKED ARRAY[0..QUESTIONS] OF ITEM;
          END;

 (* examinee personal data *)
 PINFOREC = RECORD
              LASTNAME : PACKED ARRAY[0..14] OF CHAR;
              FIRSTNAME : PACKED ARRAY[0..11] OF CHAR;
              MINITIAL : CHAR;
              CURRADDRESS,
              HOMEOFREC  : PACKED ARRAY[0..1] OF CHAR;
              CITIZENSHIP : PACKED ARRAY[0..3] OF CHAR;
              SEX : CHAR;
              POPGROUP : PACKED ARRAY[0..4] OF CHAR;
              ETHNIC : PACKED ARRAY[0..1] OF CHAR;
              MARITAL : CHAR;
              DEPENDANTS : PACKED ARRAY[0..1] OF CHAR;
              BIRTHDATE : PACKED ARRAY[0..7] OF CHAR;
              EDUCATION : PACKED ARRAY[0..2] OF CHAR;
              TESTID : PACKED ARRAY[0..2] OF CHAR;
              AFQT : PACKED ARRAY[0..1] OF CHAR;
              ASVAB  : PACKED ARRAY[0..43] OF CHAR;
              ENLISTDATE,
              ACTSERDATE  : PACKED ARRAY[0..7] OF CHAR;
              ENL : PACKED ARRAY[0..4] OF CHAR;
              AFEES : PACKED ARRAY[0..1] OF CHAR;
              SOMETHING : PACKED ARRAY[0..3] OF CHAR;
            END;
```

```
VAR MAXRECORDS,
    I : INTEGER;

    (* examinee test taking  data *)
    EXAMINEE : EXAMEINFO;
    FILEEXAMINEE : FILE OF EXAMEINFO;

    (* examinee directory *)
    DIR : INDEX;
    EDIR : FILE OF INDEX;

    (* examinee test results *)
    TESTS : SUBTEST;
    FILETESTS : FILE OF SUBTEST;

    (* examinee personal data *)
    PINFO : PINFOREC;
    PINFOFILE : FILE OF PINFOREC;



    (* format the examinee directory for the session examinees and *)
    (* the done examinee files.                                    *)
    PROCEDURE MAKEDIRECTORY;
    BEGIN
      FOR I := 0 TO MAXEXAMINEE DO
        DIR[I].UNUSED := TRUE;

      REWRITE(EDIR,EINDEX);
      SEEK(EDIR,0);
      EDIR^ := DIR;
      PUT(EDIR);
      CLOSE(EDIR,LOCK);

      REWRITE(EDIR,DONEINDEX);
      SEEK(EDIR,0);
      EDIR^ := DIR;
      PUT(EDIR);
      CLOSE(EDIR,LOCK);
    END;      (* make directory *)


    (* format the files containing the test taking information *)
    PROCEDURE MAKETINFO;
    BEGIN
      WITH EXAMINEE DO
      BEGIN
        ID := '         ';
        LASTTEST := 10; (* mark no tests taken yet *)
        NUMPROC := 0;
        TOTTIMECONSOLE := 0;
        NUMERRORS := 0;
        PREVTIMELASTTEST := 0;
        ORIENTATIONTIME := 0;
      END;

      REWRITE(FILEEXAMINEE,INFONAME);
      SEEK(FILEEXAMINEE,0);
      FOR I := 0 TO MAXEXAMINEE DO
      BEGIN
        FILEEXAMINEE^ := EXAMINEE;
        PUT(FILEEXAMINEE);
      END;
      CLOSE(FILEEXAMINEE,LOCK);

      REWRITE(FILEEXAMINEE,DONEINFO);
      SEEK(FILEEXAMINEE,0);
      FOR I := 0 TO MAXEXAMINEE DO
      BEGIN
        FILEEXAMINEE^ := EXAMINEE;
        PUT(FILEEXAMINEE);
      END;
      CLOSE(FILEEXAMINEE,LOCK);
```

```
END;    (* maketinfo *)


(*  format the files containg testing results information *)
PROCEDURE FORMATRESULTSFILE;
BEGIN
  MAXRECORDS := (MAXEXAMINEES * GMAXSUBTEST) + GMAXSUBTEST;
  WITH TESTS DO
  BEGIN
    STTIME := 0;
    STINSTRTIME := 0;
    STPROCTCALLS := 0;
    STERRKEY := 0;
    NUMITEMS := 0;
    NUMCORR := 0;
    ESTABILITY := 0;
  END;

  REWRITE(FILETESTS,RESULTS);
  SEEK(FILETESTS,0);
  WRITELN;
  FOR I := 0 TO MAXRECORDS DO
  BEGIN
    IF (I MOD 10) = 0 THEN
      WRITE('.');
    FILETESTS^ := TESTS;
    PUT(FILETESTS);
  END;
  CLOSE(FILETESTS,LOCK);

  REWRITE(FILETESTS,DONERESULTS);
  SEEK(FILETESTS,0);
  WRITELN;
  FOR I := 0 TO MAXRECORDS DO
  BEGIN
    IF (I MOD 10) = 0 THEN
      WRITE('.');
    FILETESTS^ := TESTS;
    PUT(FILETESTS);
  END;
  CLOSE(FILETESTS,LOCK);
END;      (* formatresultsfile *)


(* format the personal data file *)
PROCEDURE MAKEPERSONALINFO;
BEGIN
  WITH PINFO DO
  BEGIN
    LASTNAME := '                ';
    FIRSTNAME := '             ';
    MINITIAL := ' ';
    CURRADDRESS := '   ';
    HOMEOFREC := '   ';
    CITIZENSHIP := '     ';
    SEX := ' ';
    POPGROUP := '      ';
    ETHNIC := '   ';
    MARITAL := '  ';
    DEPENDANTS := '   ';
    BIRTHDATE := '         ';
    EDUCATION := '    ';
    TESTID := '    ';
    AFQT := '   ';
    ASVAB := '                                             ';
    ENLISTDATE := '         ';
    ACTSERDATE := '         ';
    ENL := '      ';
    AFEES := '   ';
    SOMETHING := '     ';
  END;

  REWRITE(PINFOFILE,PINFONAME);
```

```
      SEEK(PINFOFILE,0);

      FOR I := 0 TO 50 DO
      BEGIN
        PINFOFILE^ := PINFO;
        PUT(PINFOFILE);
      END;
      CLOSE(PINFOFILE,LOCK);

      REWRITE(PINFOFILE,DONEPINFO);
      SEEK(PINFOFILE,0);

      FOR I := 0 TO 50 DO
      BEGIN
        PINFOFILE^ := PINFO;
        PUT(PINFOFILE);
      END;
      CLOSE(PINFOFILE,LOCK);
    END;  (* makepersonalinfo *)


BEGIN   (* eformat *)
  PAGE(OUTPUT);
  WRITELN;
  WRITELN(' Are you sure you want');
  WRITE(' to zero the directory?  Y/N : ');
  IF GETCHAR(['Y','N'],TRUE,TRUE,TRUE) = 'Y' THEN
  BEGIN
    (* MAKEDIRECTORY; *)

    MAKETINFO;
    FORMATRESULTSFILE;
    (* MAKEPERSONALINFO; *)

  END;
END;    (* eformat *)


(* Initialize the startup format *)
PROCEDURE STARTUPFORMAT;

CONST SETUPDATA = 'CATDATA:PARAMETERS.DATA';  (* test default parameters *)

TYPE (* set-up parameters *)
     SETUPINFO = PACKED RECORD
                   SUBORDER,
                   SUBSTRAT       : PACKED ARRAY[1..GMAXSUBTEST]
                                    OF 0..128;
                   ITEMFB,
                   ITEMOUTPUT,
                   SUBTESTFB,
                   SUBTESTOUTPUT,
                   SESSIONFB,
                   SESSIONOUTPUT  : 0..128;
                   SUBSTOP        : PACKED ARRAY[1..GMAXSUBTEST] OF BOOLEAN;
                   SUBLENGTH      : PACKED ARRAY[1..GMAXSUBTEST]
                                    OF 0..128;
                   CKERROR        : ARRAY[1..GMAXSUBTEST] OF REAL;
                 END;

VAR I : INTEGER;

    (*** set-up variables ***)
    SPARAMS : SETUPINFO;
    FILESPARAMS : FILE OF SETUPINFO;

BEGIN
  PAGE(OUTPUT);
  WRITELN;
  WRITELN(' Are you sure you want');
  WRITE(' to zero the parameter file.  Y/N : ');
  IF GETCHAR(['y','Y','N','n'],TRUE,TRUE,TRUE) IN ['y','Y']  THEN
  BEGIN
    WITH SPARAMS DO
    BEGIN
      FOR I := 1 TO GMAXSUBTEST DO
      BEGIN
        SUBORDER[I] := 128;
        SUBSTRAT[I] := 0;
        SUBSTOP[I] := FALSE;
        SUBLENGTH[I] := 0;
        CKERROR[I] := 1.0;
      END;
      ITEMFB := 0;
      ITEMOUTPUT := 0;
      SUBTESTFB := 0;
      SUBTESTOUTPUT := 0;
      SESSIONFB := 0;
      SESSIONOUTPUT := 0;
    END;
    REWRITE(FILESPARAMS,SETUPDATA);
    SEEK(FILESPARAMS,0);
    FILESPARAMS^ := SPARAMS;
    PUT(FILESPARAMS);
    CLOSE(FILESPARAMS,LOCK);
  END;
END;    (* startupformat *)


(* infotable files initialization *)
PROCEDURE INFOSETUP;

CONST(* information tables *)
      TABNAME = 'CATDATA:TABINFO.DATA';


      (* information table dimensions *)
      INFOROW = 36;
      INFOCOLUMN = 20;

      MAXSUBTESTS = 20;


TYPE TABLE = ARRAY[1..INFOCOLUMN,1..INFOROW] OF INTEGER;


VAR COMMAND : CHAR;

    I,J : INTEGER;

    INFOTABLE : TABLE;
    INFOFILE  : FILE OF TABLE;


BEGIN   (* infosetup *)
  PAGE(OUTPUT);
  WRITELN('Are you sure you wish to initialize?');
  WRITELN('This will destroy all existing info-');
  WRITELN('tables.');
  IF GETCHAR(['Y','N','n','y'],TRUE,FALSE,TRUE) IN ['Y','y'] THEN
  BEGIN
    WRITELN;
    WRITE('Last chance!  Do you wish to initialize?');
    IF GETCHAR(['Y','N','n','y'],TRUE,FALSE,TRUE) IN ['Y','y'] THEN
    BEGIN
      FOR J := 1 TO INFOROW DO
        FOR I := 1 TO INFOCOLUMN DO
          INFOTABLE[I,J] := NIL;
      REWRITE(INFOFILE,TABNAME);
      PAGE(OUTPUT);
      WRITE('Initializing');
      FOR I := 0 TO ((MAXSUBTESTS * 2) + 1) DO
      BEGIN
        WRITE('.');
```

```
             SEEK(INFOFILE,I);
             INFOFILE^ := INFOTABLE;
             PUT(INFOFILE);
           END;
           CLOSE(INFOFILE,LOCK);  .
         END;
       END;
     END; (* initfile *)


     (* give warning message *)
     PROCEDURE WARNING;
     var chr : char;
     BEGIN
       PAGE(OUTPUT);
       WRITELN('             ****************** WARNING ******************');
       writeln;
       writeln('                  This is a very dangerous program so');
       writeln('              if you dont know what you are doing, get');
       writeln('              out now while you still can with your');
       writeln('              life.  If you execute any of the options');
       writeln('              in this program, you will anihilate all');
       writeln('              data in certain files and cause hours of');
       writeln('              hard work to be lost.');
       writeln;
       WRITELN;
       WRITELN;
       write(' DO YOU WISH TO CONTINUE ?    : ');
       readln(chr);
       if not (chr in ['y','Y']) then
         exit(program);
       writeln;
       write(' ARE YOU SURE YOU WANT TO CONTINUE ?   : ');
       readln(chr);
       if not (chr in ['y','Y']) then
         exit(program);
       writeln;
       write(' LAST CHANCE !!!!!    CONTINUE ?   : ');
       readln(chr);
       if not (chr in ['y','Y']) then
         exit(program);
     END;   (* warning *)

(* main program *)

BEGIN
  WARNING;
  REPEAT
    PAGE(OUTPUT);
    GOTOXY(5,5);
    WRITE('1) Format Subtest Database Files');
    GOTOXY(5,7);
    WRITE('2) Format Examinee Database Files');
    GOTOXY(5,9);
    WRITE('3) Format System Parameter Files');
    GOTOXY(5,11);
    WRITE('4) Format Infotables');
    GOTOXY(5,13);
    WRITE('5) Quit');
    GOTOXY(1,1);
    WRITE('Command : ');
    COMMAND := GETCHAR(['1'..'4'],TRUE,TRUE,TRUE);
    CASE COMMAND OF
      '1' : (* SUBTESTFORMAT *);
      '2' : EFORMAT;
      '3' : STARTUPFORMAT;
      '4' : INFOSETUP;
      '5' : ;
    END;
  UNTIL COMMAND = '5';
 END.   (* format files *)
```

```
(*$S+*)
(******************************************************************************)
(*                                                                            *)
(*      Textfile : MISC/CATKRUNCH.TEXT          Volume : TFILES               *)
(*      Codefile :                              Volume :                      *)
(*                                                                            *)
(******************************************************************************)
(*              DEC.  1, 1982                   NPROC                         *)
(******************************************************************************)
(*                                                                            *)
(* This program goes through all the subtests and krunches the unused space   *)
(* between text in the ascii file and resets the block and byte pointers for  *)
(* each question.                                                             *)
(*                                                                            *)
(******************************************************************************)
(*                                                                            *)
(* The old files, directory data, and text are not changed, in case of        *)
(* program crash during krunch.  Instead, 3 new files are created.  They are: *)
(*                                                                            *)
(*                      1. Newdir.data                                        *)
(*                      2. Newdata.data                                       *)
(*                      3. Newtext.data                                       *)
(*                                                                            *)
(* These are respectively, the directory, data and textfiles.  They are       *)
(* written to the same volumes the old files reside on.  To use the new       *)
(* files, change the filenames declared in each program, test to see if       *)
(* any data errors occured in transfer, then change the new file names to     *)
(* the old file names on the volumes and in the programs, so the next future  *)
(* krunch will see the new files as the old files.                            *)
(*                                                                            *)
(******************************************************************************)


PROGRAM TEXTTRANSFER;
CONST (* block sized buffer for ascii *)
      MAXITEMBUF = 511;
      NIL = -1;


      (* question textfile control codes *)
      GOTOFLAG = 128;       (* flags a gotoxy *)
      PAGEFLAG = 129;       (* flags text continues on another page *)
      UNUSEDFLAG = 130;     (* flags unused byte *)
      ENDITEM = 131;        (* flags end of text for a question *)

      (* these files must reside on disk ! *)
      DATANAME = 'CATDATA:ITEMPOOL.DATA';    (* question data *)
      TEXTNAME = 'QTEXT:ITEMTEXT.DATA';    (* question text *)
      INDEXNAME = 'CATDATA:TESTINDEX.DATA'; (* test directory *)


      (* slots available in directory *)
      MAXSUBTESTS = 20;
      (* maximum question pool per test *)
      MAXITEMPOOL = 300;

      (* maximum # of sample questions allowed *)
      MAXSAMPLES = 5;


TYPE DIRDATA = PACKED RECORD  (* directory for tests *)
                    UNUSED : BOOLEAN;    (* tells if record occupied *)
                    TESTNAME : STRING;  (* name of subtest *)
                    (* subtest directory of question id codes *)
                    ITEMCODE : PACKED ARRAY
                              [0..MAXITEMPOOL]
                              OF INTEGER;
                 END;

     (* types of answers to questions *)
     ITEMRESPONSES = (CHARVALUE,INTVALUE,SEVENCHR);
     ITEMDATA = PACKED RECORD  (* question ptrs/data *)

                          (* flag this as graphics item *)
                          GRAPHICS : BOOLEAN;
                          (* bounds for response range *)
                          LOWANSWER,
                          HIGHANSWER : CHAR;
                          (* block # where question text starts *)
                          ITEMBLOCK,
                          (* byte ptr where question text starts *)
                          ITEMPTR,
                          (* # of answers for multiple question screens *)
                          ANSWERCOUNT : INTEGER;
                          (* information parameters *)
                          A,B,C,
                          PROPCORRECT,
                          POINTBISERIAL,
                          YOPT,
                          XOPT,
                          DUMMY1,
                          DUMMY2,
                          DUMMY3  : REAL;

                          (* answer to question *)
                          CASE ATYPE : ITEMRESPONSES OF
                            CHARVALUE : (ANSWER : CHAR);
                            INTVALUE  : (INTANSWER : INTEGER);
                            SEVENCHR  : (CHRANSWER : PACKED ARRAY[1..7] OF
                                                     CHAR);
                        END;


VAR ITEMBUF : PACKED ARRAY[0..511] OF 0..139;

    CURRINDEXRECNUM,  (* record # of file directory *)
    CURRBLOCK,  (* block # of start of item text *)
    CURRFREEPTR : INTEGER; (* ptr to free loc in block *)


    DIRECTORY : DIRDATA;
    FILEDIRECTORY : FILE OF DIRDATA;

    ITEMINFO : ITEMDATA;
    FILEITEMINFO : FILE OF ITEMDATA;

    (* file of ascii codes, control #'s *)
    ITEMTEXT : FILE;


PROCEDURE STALL;
BEGIN
  WRITE('Type return');
  readln;
end;


(* returns the slot # in subtest directory where question is, *)
(* nil if the question code is not in the directory           *)
FUNCTION SLOTSEARCH(CODE : INTEGER) : INTEGER;
VAR SLOT : INTEGER;
    FOUND : BOOLEAN;
BEGIN
  SLOT := MAXSAMPLES + 1;
  FOUND := FALSE;
  REPEAT
    IF DIRECTORY.ITEMCODE[SLOT] = CODE THEN
      FOUND := TRUE
    ELSE
      SLOT := SLOT + 1;
  UNTIL (SLOT > MAXITEMPOOL) OR (FOUND);
  IF FOUND THEN
    SLOTSEARCH := SLOT
  ELSE
    SLOTSEARCH := NIL;
END;  (* slot search *)



(* returns record # of question data file *)
(* no collisions.  This is a mapping      *)
(* function which takes the location a    *)
(* question code exists in a subtest      *)
(* directory, the maximum questions per   *)
(* subtest and the subtest record number  *)
(* and maps it to a location in a file    *)
(* with data for that question            *)
FUNCTION HASH(KEY : INTEGER) : INTEGER;
BEGIN
  HASH :=
    (CURRINDEXRECNUM * MAXITEMPOOL)
    + KEY + CURRINDEXRECNUM;
END;  (* hash *)


(* reads a block from disk into the item ascii buffer *)
PROCEDURE READITEMBLOCK(WHICHBLOCK : INTEGER);
VAR BLOCKSTRANSFERRED,
    ERRNUM : INTEGER;

    BADIO : BOOLEAN;
BEGIN
  BADIO := FALSE;
  RESET(ITEMTEXT,TEXTNAME);  (* question text *)
  BLOCKSTRANSFERRED :=
    BLOCKREAD(ITEMTEXT,ITEMBUF,1,WHICHBLOCK);
  BADIO := ((BLOCKSTRANSFERRED < 1) OR (IORESULT <> 0));
  ERRNUM := IORESULT;
  CLOSE(ITEMTEXT,LOCK);
  IF BADIO THEN
  BEGIN
    WRITELN;WRITELN;
    WRITE('Block read io error # ',ERRNUM);
    STALL;
    EXIT(PROGRAM);
  END;
END;


PROCEDURE TRANSFER;
CONST FDIRNAME = 'CATDATA:NEWDIR.DATA';
      FDATANAME = 'CATDATA:NEWDATA.DATA';
      FTEXTNAME = 'QTEXT:NEWTEXT.DATA';


VAR I,
    J,
    K,
    DATARECNUM,
    FCURRBLOCK,                (* current free block *)
    FCURRFREEPTR,
    ERRNUM : INTEGER;     (* current free byte in free block *)

    (* directory *)
    FDIR           : DIRDATA;
    FDIRFILE       : FILE OF DIRDATA;

    (* question data *)
    FDATA          : ITEMDATA;
    FDATAFILE      : FILE OF ITEMDATA;

    (* question text *)
    FTEXT          : FILE;
    (* block sized buffer for ascii & control codes *)
    FITEMBUF         : PACKED ARRAY[0..511] OF 0..139;


     (* This procedure writes a block of question ascii from the block *)
     (* sized buffer to the disk in the ascii file.                    *)
     PROCEDURE FWRITEITEMBLOCK(WHICHBLOCK : INTEGER);
     VAR BLOCKSTRANSFERRED,
         ERRNUM : INTEGER;
         BADIO : BOOLEAN;
     BEGIN
       BADIO := FALSE;
       RESET(FTEXT,FTEXTNAME);
       BLOCKSTRANSFERRED :=
         BLOCKWRITE(FTEXT,FITEMBUF,1,WHICHBLOCK);
       BADIO := ((BLOCKSTRANSFERRED < 1) OR (IORESULT <> 0));
       ERRNUM := IORESULT;
       CLOSE(FTEXT,LOCK);
       IF BADIO THEN
       BEGIN
         WRITELN;WRITELN;
         WRITE('Block write io error # ',ERRNUM);
         WRITELN;
         STALL;
         EXIT(PROGRAM);
       END;
     END;  (* fwriteitemblock *)


     (* reads a block from disk into the item ascii buffer *)
     PROCEDURE FREADITEMBLOCK(WHICHBLOCK : INTEGER);
     VAR BLOCKSTRANSFERRED,
         ERRNUM : INTEGER;
         BADIO : BOOLEAN;
     BEGIN
       BADIO := FALSE;
       RESET(FTEXT,FTEXTNAME);
       BLOCKSTRANSFERRED :=
         BLOCKREAD(FTEXT,FITEMBUF,1,WHICHBLOCK);
       BADIO := ((BLOCKSTRANSFERRED < 1) OR (IORESULT <> 0));
       ERRNUM := IORESULT;
       CLOSE(FTEXT,LOCK);
       IF BADIO THEN
       BEGIN
         WRITELN;WRITELN;
         WRITEln('Block read io error # ',ERRNUM);
         WRITELN;
         STALL;
         EXIT(PROGRAM);
       END;
     END;


     (* saves value of free space, block & byte ptr*)
     (* in block 0, bytes 0..3 of text file        *)
     PROCEDURE FSAVEPTRS;
     VAR TRIX : RECORD CASE INTEGER OF
                  1 : (TWOBYTES : PACKED ARRAY
                                    [0..1] OF CHAR);
                  2 : (INTVALUE : INTEGER);
                END;
     BEGIN
       FREADITEMBLOCK(0);
       TRIX.INTVALUE := FCURRBLOCK;
       MOVELEFT(TRIX.TWOBYTES[0],FITEMBUF[0],2);
       TRIX.INTVALUE := FCURRFREEPTR;
       MOVELEFT(TRIX.TWOBYTES[0],FITEMBUF[2],2);
       FWRITEITEMBLOCK(0);
     END;   (* fsave ptrs *)
     (* This procedure reads in the question text from the ascii file *)
     (* of the corvus and block reads it onto the floppy.             *)
     PROCEDURE ASCIITODISK(CBLOCK,CPTR : INTEGER);
     VAR CORVUSBLOCK,
         CORVUSPTR,
         CHARCOUNT    : INTEGER;
     BEGIN
       CORVUSBLOCK := CBLOCK;
       CORVUSPTR := CPTR;
       READITEMBLOCK(CORVUSBLOCK);     (* get block where text starts *)
       FREADITEMBLOCK(FCURRBLOCK);     (* set buffer to fill *)


       (* read from corvus buffer into floppy buffer, if ptrs reach end *)
       (* of buffer, read in new block/write out full buffer            *)
       WHILE ITEMBUF[CORVUSPTR] <> ENDITEM DO
       BEGIN
         FITEMBUF[FCURRFREEPTR] := ITEMBUF[CORVUSPTR];
         FCURRFREEPTR := FCURRFREEPTR + 1;
         CORVUSPTR := CORVUSPTR + 1;
         IF CORVUSPTR > 511 THEN
         BEGIN
           CORVUSBLOCK := CORVUSBLOCK + 1;
           CORVUSPTR := 0;
           READITEMBLOCK(CORVUSBLOCK);
         END;
         IF FCURRFREEPTR > 511 THEN
         BEGIN
           FWRITEITEMBLOCK(FCURRBLOCK);
           FCURRBLOCK := FCURRBLOCK + 1;
           FCURRFREEPTR := 0;
         END;
       END;
       FITEMBUF[FCURRFREEPTR] := ENDITEM;    (* mark end of text *)
       FCURRFREEPTR := FCURRFREEPTR + 1;
       FWRITEITEMBLOCK(FCURRBLOCK);
       IF FCURRFREEPTR > 511 THEN
       BEGIN
         FCURRBLOCK := FCURRBLOCK + 1;
         FCURRFREEPTR := 0;
         FWRITEITEMBLOCK(FCURRBLOCK);
       END;
     END;   (* ascii to floppy *)


BEGIN    (* transferto5inch *)
  FCURRBLOCK := 0;
  FCURRFREEPTR := 4;     (* first four bytes reserved 0 - 3 *)

  REWRITE(FTEXT,FTEXTNAME);
  ERRNUM := BLOCKWRITE(FTEXT,FITEMBUF,1,0);
  CLOSE(FTEXT,LOCK);


  FOR K := 0 TO MAXSUBTESTS DO
  BEGIN

    (* get the subtest directory *)
    RESET(FILEDIRECTORY,INDEXNAME);
    SEEK(FILEDIRECTORY,K);
    GET(FILEDIRECTORY);
    DIRECTORY := FILEDIRECTORY^;
    CLOSE(FILEDIRECTORY,NORMAL);
    FDIR := DIRECTORY;

    CURRINDEXRECNUM := K;


    IF NOT (DIRECTORY.UNUSED) THEN
    BEGIN
      RESET(FILEITEMINFO,DATANAME);
      RESET(FDATAFILE,FDATANAME);
      (* transfer data, text and update text pointers *)
      WRITELN;
      WRITE('Transferring ',DIRECTORY.TESTNAME);
      FOR I := 0 TO MAXITEMPOOL DO
      BEGIN
        IF DIRECTORY.ITEMCODE[I] >= 0 THEN  (* question exists *)
        BEGIN
          WRITE('.');
          DATARECNUM := HASH(I);
          SEEK(FILEITEMINFO,DATARECNUM);
          GET(FILEITEMINFO);
          ITEMINFO := FILEITEMINFO^;
          FDATA := ITEMINFO;          (* transfer the data *)
          FDATA.ITEMBLOCK := FCURRBLOCK;   (* set new text ptrs *)
          FDATA.ITEMPTR := FCURRFREEPTR;
          SEEK(FDATAFILE,DATARECNUM);
          FDATAFILE^ := FDATA;
          PUT(FDATAFILE);                  (* write data to floppy *)

          (* transfer the text *)
          ASCIITODISK(ITEMINFO.ITEMBLOCK,ITEMINFO.ITEMPTR);
        END;
      END;
      FSAVEPTRS;                (* save end of file marker *)
      CLOSE(FILEITEMINFO,LOCK);
      CLOSE(FDATAFILE,LOCK);
    END;
  END;
END;   (* transfertoSinch *)



(* main program *)

BEGIN
  TRANSFER;
END.
```